Automated Infrastructure Deployment with Terraform and GitHub Actions

Training Report

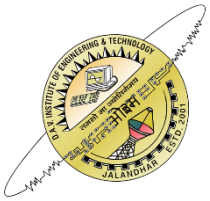
SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR SIX MONTH INDUSTRIAL TRAINING

At

cloudEQ

SUBMITTED BY

Name	:	YUVRAJ KUMAR ARORA
Branch	:	ELECTRONICS & COMMUNICATION Engg.
Roll No.	:	256/19
Univ. Roll No.	:	1903811



DAVIET

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

DAV INSTITUTE OF ENGINEERING & TECHNOLOGY

Jalandhar, India

Abstract

Automated Infrastructure Deployment with Terraform and GitHub Actions is a comprehensive project that aims to revolutionize the way infrastructure is provisioned and managed. By harnessing the power of infrastructure as code and leveraging the automation capabilities of GitHub Actions, this project provides a robust and efficient solution for continuous integration and deployment. The project, aptly named TerraDeploy, revolves around the use of Terraform, a popular and widely adopted infrastructure provisioning tool. Terraform allows users to define their infrastructure resources in a declarative manner using a simple and intuitive syntax. By treating infrastructure as code, users can version control their configurations, enabling easy collaboration, tracking of changes, and reproducibility. GitHub Actions, a powerful automation platform, serves as the backbone of the deployment pipeline in TerraDeploy. It enables users to define workflows that automate various tasks, including infrastructure testing, validation, and deployment. With GitHub Actions, users can trigger workflows based on events such as code pushes, pull requests, or scheduled intervals, ensuring continuous integration and deployment of infrastructure changes.

TerraDeploy offers a range of features designed to streamline the infrastructure deployment process. Automated testing and validation can be implemented using Terraform validation tools or custom scripts, ensuring that infrastructure configurations adhere to best practices and avoid common pitfalls. GitHub Actions workflows provide the flexibility to define separate deployment pipelines for different environments, such as development, staging, and production, allowing for controlled and standardized deployments across the project lifecycle. To maintain security and protect sensitive information, TerraDeploy integrates with GitHub Secrets, allowing users to securely store and manage credentials and API keys. This ensures that sensitive data is not exposed within the infrastructure code or in public repositories.

The project also emphasizes transparency and collaboration. By hosting the infrastructure code on GitHub, team members can easily review, discuss, and provide feedback on proposed changes. Additionally, notifications and reporting can be set up to keep stakeholders informed about the status of deployments and any issues that may arise. TerraDeploy's benefits include increased efficiency, reduced manual effort, and improved infrastructure quality. By automating the deployment process, developers and system administrators can focus on higher-level tasks and avoid repetitive and error-prone manual procedures. The use of infrastructure as code enables scalability and flexibility, allowing for easy modification and scaling of resources as project requirements evolve.

In conclusion, Automated Infrastructure Deployment with Terraform and GitHub Actions, through the TerraDeploy project, provides a comprehensive solution for infrastructure provisioning and management. It combines the power of Terraform's infrastructure as code approach with the automation capabilities of GitHub Actions, enabling continuous integration and deployment. With TerraDeploy, users can achieve efficient, scalable, and reliable infrastructure deployments while promoting collaboration and transparency within their teams.

ACKNOWLEDGEMENT

I am highly grateful to the Dr. Sanjeev Naval, Principal, DAV Institute of Engineering & Technology, Jalandhar, for providing this opportunity to carry out the six-month industrial training at CloudEQ Software India Pvt Ltd.

The constant guidance and encouragement received from Dr. Neeru Malhotra, HoD Department of Electronics & Communication Engineering, DAVIET Jalandhar has been of great help in carrying out the project work and is acknowledged with reverential thanks.

I would like to express my gratitude to Dr. Love Kumar, Assistant Professor Department of Electronics & Communication Engineering DAVIET Jalandhar for his stimulating guidance, continuous encouragement and supervision throughout the course of present work.

I would like to express a deep sense of gratitude and thanks profusely to Mr. Sean Barker CEO of Company, Without the wise counsel and able guidance, it would have been impossible to complete the report in this manner.

I would like to express a deep sense of gratitude and thanks profusely to Mr. Pradyumna Kishor Parida, Designation (Project Management in company CloudEQ Software India Pvt Ltd.) for experimentation is greatly acknowledged.

The help rendered by Mr. Pinkesh, Designation (Software Engineer in company CloudEQ Software India Pvt Ltd.) for experimentation is greatly acknowledged.

I express gratitude to other faculty members of Electronics & Communication Engineering department of DAVIET for their intellectual support throughout the course of this work.

Yuvraj Kumar Arora

LIST OF FIGURES

TABLE OF CONTENTS

Contents **Page No.**

Chapter 1: Introduction

1.1 Company Overview

cloudEQ is a professional services company with hundreds of certified experts in Microsoft, Amazon, Google, New Relic, ServiceNow, and much more. We are Fortune 100 executives, experienced leaders, and technical experts with a mission to provide experience-based cloud services. With experience on both sides of the table, cloudEQ offers a depth and breadth of knowledge you can leverage as your own. When it comes to our teams, we maintain an in-house training and learning center to ensure we're always learning and building the right team for you.

As the people who power your digital transformation, we embody the EQ (emotional intelligence) needed for a strong rapport and partnership. We demonstrate this through empathy and understanding of your business goals and your people. It's our depth of experience that leads to trust, and it's the trust in our people that leads to powerful results.

We embody the (Emotional Quotient) emotional intelligence needed for a strong rapport and partnership by truly caring about our clients and their business like no other. We demonstrate this through empathy and understanding of their business goals and their people. It's our depth of experience that leads to trust and it's the trust in our people that leads to powerful results.

1.2 Our Services

- **Cloud Migration:** Whether you're migrating for the first time or looking to expand your cloud presence, our focused team of cloud experts have the tools, resources, and know-how you need for every stage of your cloud migration strategy.

- **Application Development:** Application development requires a holistic strategy. By using a wide range of cloud capabilities, your applications and workloads can reach virtualized environments or be refactored and re-architectured for more efficiency. So, whether you're looking for enterprise DevOps, PaaS, SaaS, or microservices, you'll get extensive cloud development solutions and cloud strategies based on the best cloud provider for your business.

- **DevOps:** Measure your current software development and operational maturity and develop a strategy for where and how to apply DevOps approaches to accelerate your application delivery and improve your operational reliability and security posture. A typical DevOps model works by ensuring that development and operations teams are no longer partitioned. In fact, usually, these two teams are merged into one, where the engineers work across the entire life cycle of the application, from development and test to deployment and operations.

- **Automation:** Our team of focused automation experts leverage cutting-edge infrastructure services to automate, optimize, and manage your IT infrastructure, all while reducing your cost of compliance and IT management risks. Once you're automated, we monitor unused resources and optimize your operations to provide seamless continuity.

- **Cloud Security:** Keep your applications, systems, and data safe with a customized security strategy. At cloudEQ, you're protected with cutting-edge Identity Assurance Management and Intrusion Detection Systems. And these ironclad practices are cost-effective and custom-designed to fit your cloud environment.

- **DevSecOps:** The purpose and intent of DevSecOps is to build on the mindset that "everyone is responsible for security" with the goal of safely distributing security decisions at speed and scale to those who hold the highest level of context without sacrificing the safety required. DevSecOps is expanding the Dev + Ops collaboration to include Security.
- **Operations:** Day-to-day management of cloud, application, data and supporting systems through standard ITOM / ITSM and SLA-based services. We run some of the world's largest footprints for fortune 100 companies. We have experience and expertise in Azure, AWS, GCP, ServiceNow, Jira, and all types of applications support. We have, or you can provide, the tools required to run or help you manage your cloud, application, and data operations.

And other Services are Cloud Optimization, Transformation, Major Incident Management (MIM), Managed Services, Project Services etc.

1.3 Our Technology Partnerships

Technology Partnerships

Names you know, service you can count on.

Figure 1: Names of our Technology Partnerships.

723

Certifications

Chapter 2: Internship

1. DEVOPS Tool Sessions

In which Session till date I have learn lots of new tools.

Terraform **GitHub – Action** **Docker**

Figure 2: Figures of new tools (DEVOPS Tool Sessions) which learn in this session.

2. PROGRAMMING Sessions

In which Session till date I have learn lots of new things.

Python **Rest API** **Bash**

Figure 3: Figures of new tools (PROGRAMMING Sessions) which learn in this session.

3. SYSOPS Sessions

In which Session till date I have learn lots of new things.

Amazon Web Services **Microsoft Azure**

Figure 4: Figures of new tools (SYSOPS Sessions) which learn in this session.

DEVOPS Tools Sessions

2.1 DevOps

DevOps is a combination of software development (dev) and operations (ops). It is defined as a software engineering methodology which aims to integrate the work of development teams and operations teams by facilitating a culture of collaboration and shared responsibility.

DevOps combines development and operations to increase the efficiency, speed, and security of software development and delivery compared to traditional processes.

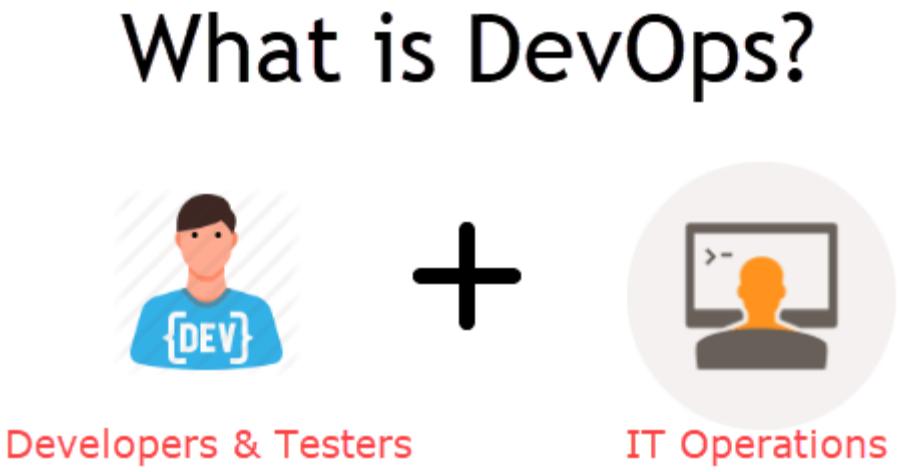


Figure 5: DevOps Explanation

Figure 6: Describe the DevOps lifecycle.

2.2 Terraform

Terraform is an infrastructure as code tool that lets you build, change, and version cloud and on-prem resources safely and efficiently.

HashiCorp Terraform is an infrastructure as code tool that lets you define both cloud and on-prem resources in human-readable configuration files that you can version, reuse, and share. You can then use a consistent workflow to provision and manage all your infrastructure throughout its lifecycle [1]. Terraform can manage low-level components like compute, storage, and networking resources, as well as high-level components like DNS entries and SaaS features.

How does Terraform work?

Terraform creates and manages resources on cloud platforms and other services through their application programming interfaces (APIs). Providers enable Terraform to work with virtually any platform or service with an accessible API.

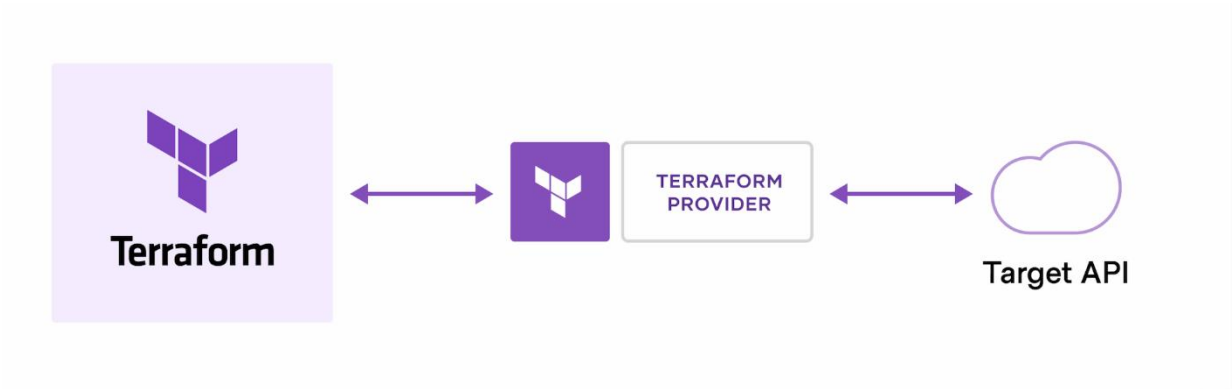


Figure 7: This figure explains how Terraform works

HashiCorp and the Terraform community have already written **thousands of providers** to manage many different types of resources and services [2]. You can find all publicly available providers on the Terraform Registry, including Amazon Web Services (AWS), Azure, Google Cloud Platform (GCP), Kubernetes, Helm, GitHub, Splunk, DataDog, and many more.

The core Terraform workflow consists of three stages:

- **Write:** You define resources, which may be across multiple cloud providers and services. For example, you might create a configuration to deploy an application on virtual machines in a Virtual Private Cloud (VPC) network with security groups and a load balancer.
- **Plan:** Terraform creates an execution plan describing the infrastructure it will create, update, or destroy based on the existing infrastructure and your configuration.
- **Apply:** On approval, Terraform performs the proposed operations in the correct order, respecting any resource dependencies. For example, if you update the properties of a VPC and change the number of virtual machines in that VPC, Terraform will recreate the VPC before scaling the virtual machines.

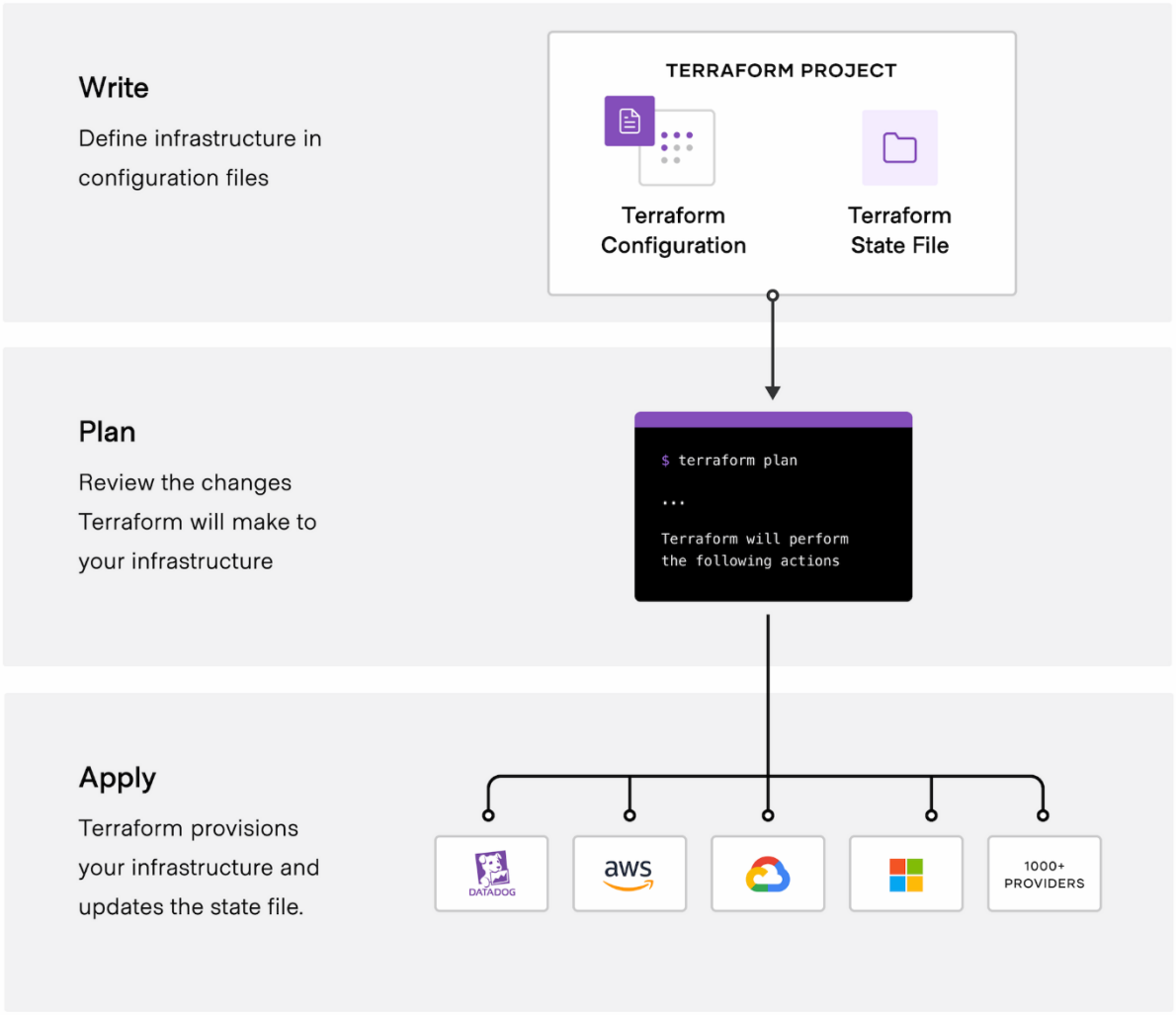


Figure 8: This figure explains terraform workflow consist of three stages.

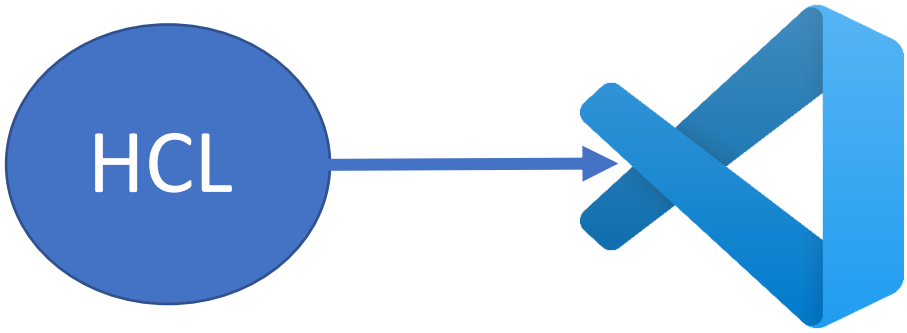


Figure 9: This Figure explains run the terraform script in a Visual Studio.

Basic Commands:

1. terraform init

The terraform init command initializes a working directory containing Terraform configuration files. This is the first command that should be run after writing a new Terraform configuration or cloning an existing one from version control. It is safe to run this command multiple times.

2. terraform validate

The terraform validate command validates the configuration files in a directory, referring only to the configuration and not accessing any remote services such as remote state, provider APIs, etc.

3. terraform fmt

The terraform fmt command is used to rewrite Terraform configuration files to a canonical format and style. This command applies a subset of the Terraform language style conventions, along with other minor adjustments for readability.

4. terraform plan

The terraform plan command lets you to preview the actions Terraform would take to modify your infrastructure or save a speculative plan which you can apply later. The function of terraform plan is speculative: you cannot apply it unless you save its contents and pass them to a terraform apply command.

5. terraform apply

The terraform apply command performs a plan just like terraform plan does, but then actually carries out the planned changes to each resource using the relevant infrastructure provider's API. It asks for confirmation from the user before making any changes, unless it was explicitly told to skip approval.

6. terraform destroy

The terraform destroy command terminates resources managed by your Terraform project. This command is the inverse of terraform apply in that it terminates all the resources specified in your Terraform state. It does not destroy resources running elsewhere that are not managed by the current Terraform project.

2.3 CI / CD Pipeline

CI / CD Pipeline, firstly we introduced CI (**Continuous Integration**) is a process in DevOps where changes are merged into a central repository after which the code is automated and tested. The continuous integration process is a practice in software engineering used to merge developers' working copies several times a day into a shared mainline. Continuous Integration is a software development practice that integrates code into a shared repository frequently [3]. This is done by developers several times a day each time they update the codebase. Each of these integrations can then be tested automatically.

Now, we introduced CD (**Continuous delivery, and continuous deployment**) **Continuous delivery** (CD) is the process of automating build, test, configuration, and deployment from a build to a production environment. A release pipeline can create multiple testing or staging environments to automate infrastructure creation and deploy new builds.

Continuous deployment (CD) is a strategy in software development where code changes to an application are released automatically into the production environment. This automation is driven by a series of predefined tests. Once new updates pass those tests, the system pushes the updates directly to the software's users.

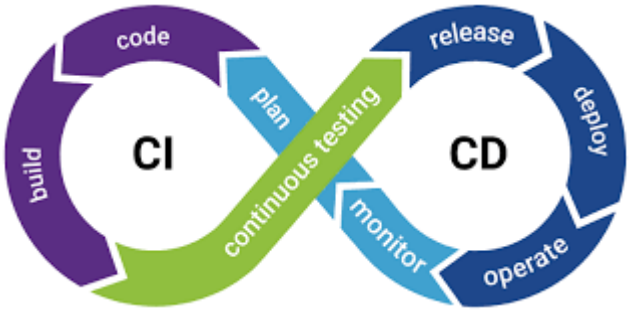


Figure 10: This figure tells us about CI / CD Pipeline.

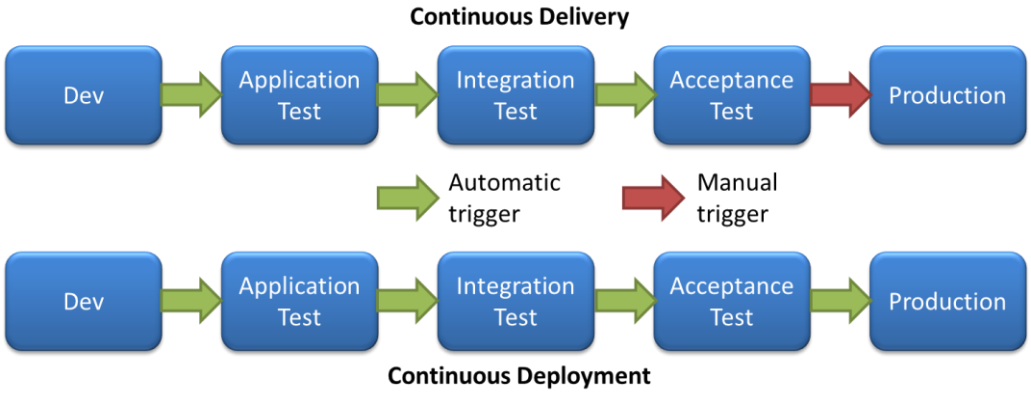


Figure 11: This Figure tells us about difference between Continuous delivery, and continuous deployment.

2.4 GitHub – Actions

GitHub – Actions is a tool of continuous integration and continuous delivery (CI/CD) platform that allows you to automate your build, test, and deployment pipeline. You can create workflows that build and test every pull request to your repository or deploy merged pull requests to production. GitHub Actions goes beyond just DevOps and lets you run workflows when other events happen in your repository [4]. For example, you can run a workflow to automatically add the appropriate labels whenever someone creates a new issue in your repository.

GitHub provides Linux, Windows, and macOS virtual machines to run your workflows, or you can host your own self-hosted runners in your own data center or cloud infrastructure.

The Components of GitHub – Actions

You can configure a GitHub Actions *workflow* to be triggered when an *event* occurs in your repository, such as a pull request being opened, or an issue being created. Your workflow contains one or more *jobs* which can run in sequential order or in parallel. Each job will run inside its own virtual machine *runner*, or inside a container, and has one or more *steps* that either run a script that you define or run an *action*, which is a reusable extension that can simplify your workflow.

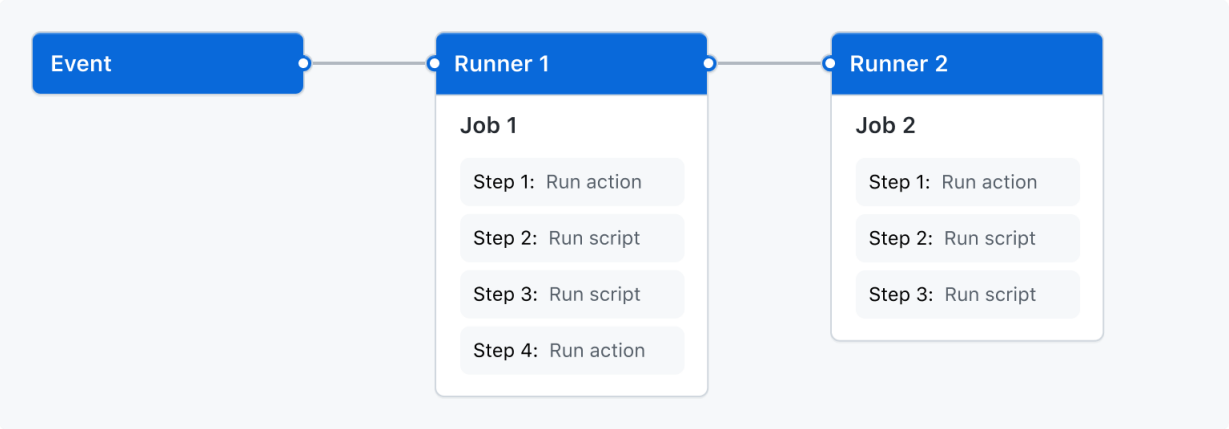


Figure 12: This Figure tells us about how GitHub – Actions does work.

Workflow A workflow is a configurable automated process that will run one or more jobs. Workflows are defined by a YAML file checked in to your repository and will run when triggered by an event in your repository, or they can be triggered manually, or at a defined schedule.

Workflows are defined in the .github/workflows directory in a repository, and a repository can have multiple workflows, each of which can perform a different set of tasks. For example, you can have one workflow to build and test pull requests, another workflow to deploy your application every time a release is created, and still another workflow that adds a label every time someone opens a new issue.

Events An event is a specific activity in a repository that triggers a workflow run. For example, activity can originate from GitHub when someone creates a pull request, opens an issue, or pushes a commit to a repository. You can also trigger a workflow to run on a schedule, by posting to a REST API or manually.

Jobs A job is a set of *steps* in a workflow that is executed on the same runner. Each step is either a shell script that will be executed, or an *action* that will be run. Steps are executed in order and are dependent on each other. Since each step is executed on the same runner, you can share data from one step to another. For example, you can have a step that builds your application followed by a step that tests the application that was built.

You can configure a job's dependencies with other jobs; by default, jobs have no dependencies and run in parallel with each other. When a job takes a dependency on another job, it will wait for the dependent job to complete before it can run. For example, you may have multiple build jobs for different architectures that have no dependencies, and a packaging job that is dependent on those jobs. The build jobs will run in parallel, and when they have all completed successfully, the packaging job will run.

Actions An *action* is a custom application for the GitHub Actions platform that performs a complex but frequently repeated task. Use an action to help reduce the amount of repetitive code that you write in your workflow files. An action can pull your git repository from GitHub, set up the correct toolchain for your build environment, or set up the authentication to your cloud provider.

You can write your own actions, or you can find actions to use in your workflows in the GitHub Marketplace.

Runners A runner is a server that runs your workflows when they're triggered. Each runner can run a single job at a time. GitHub provides Ubuntu Linux, Microsoft Windows, and macOS runners to run your workflows; each workflow run executes in a fresh, newly provisioned virtual machine.

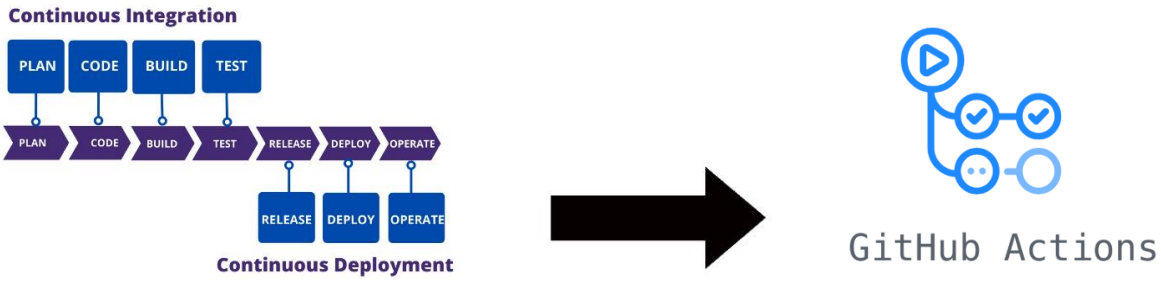


Figure 13: Full-fledged figure describes the CI / CD pipeline and GitHub – Actions.

2.5 Docker or Docker Engine

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

Docker provides the ability to package and run an application in a loosely isolated environment called a container. The isolation and security allow you to run many containers simultaneously on a given host. Containers are lightweight and contain everything needed to run the application, so you do not need to rely on what is currently installed on the host [5]. You can easily share containers while you work and be sure that everyone you share with gets the same container that works in the same way.

Docker provides tooling and a platform to manage the lifecycle of your containers:

- Develop your application and its supporting components using containers.
- The container becomes the unit for distributing and testing your application.
- When you're ready, deploy your application into your production environment, as a container or an orchestrated service. This works the same whether your production environment is a local data center, a cloud provider, or a hybrid of the two.

Docker Architecture Docker uses a client-server architecture. The Docker *client* talks to the Docker *daemon*, which does the heavy lifting of building, running, and distributing your Docker containers. The Docker client and daemon *can* run on the same system, or you can connect a Docker client to a remote Docker daemon. The Docker client and daemon communicate using a REST API, over UNIX sockets or a network interface. Another Docker client is Docker Compose, that lets you work with applications consisting of a set of containers.

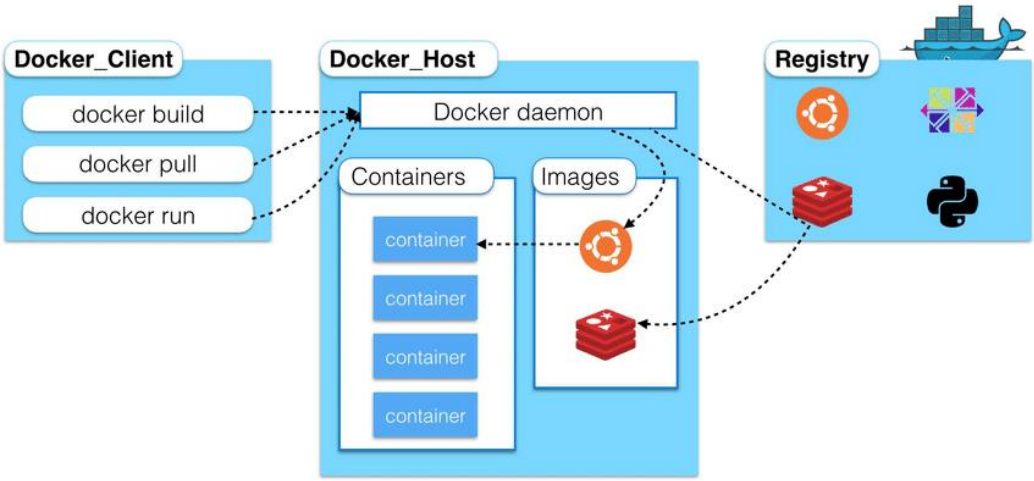


Figure 14. This figure tells us about the Docker Architecture.

Docker Object When you use Docker, you are creating and using images, containers, networks, volumes, plugins, and other objects. This section is a brief overview of some of those objects.

Images

An *image* is a read-only template with instructions for creating a Docker container. Often, an image is *based on* another image, with some additional customization. For example, you may build an image which is based on the ubuntu image, but installs the Apache web server and your application, as well as the configuration details needed to make your application run.

You might create your own images, or you might only use those created by others and published in a registry. To build your own image, you create a *Dockerfile* with a simple syntax for defining the steps needed to create the image and run it. Each instruction in a Dockerfile creates a layer in the image. When you change the Dockerfile and rebuild the image, only those layers which have changed are rebuilt. This is part of what makes images so lightweight, small, and fast, when compared to other virtualization technologies.

Containers

A container is a runnable instance of an image. You can create, start, stop, move, or delete a container using the Docker API or CLI. You can connect a container to one or more networks, attach storage to it, or even create a new image based on its current state.

By default, a container is relatively well isolated from other containers and its host machine. You can control how isolated a container's network, storage, or other underlying subsystems are from other containers or from the host machine.

A container is defined by its image as well as any configuration options you provide to it when you create or start it. When a container is removed, any changes to its state that are not stored in persistent storage disappear.

Basic Commands:

1. **docker build -t "tag_name" .** (this command to build the image)
2. **docker images** (this command to check all the images)
3. **docker run -d -p 8002:8000 tag_name** (this command to build the container in which 8002 port is take any port to run and 8000 port is fixed port to expose Django application)
4. **docker ps** (this command to check the all running containers)
5. **docker ps -a** (this command to check all running and not running containers)
6. **docker stop container_Id** (this command is used to stop the container)
7. **docker start container_Id** (this command is used to start again container)
8. **docker rm container_id** (this command is used to delete the permanently container)
9. **docker rmi image_Id** (this command is used to delete the image)

Note: Once you stop the container then if you delete the image you cannot start again that container. If you, have create again that image then you must build again that image.

2.6 Cloud Computing

Cloud Computing Cloud computing is a virtualization-based technology that allows us to create, configure, and customize applications via an internet connection. The cloud technology includes a development platform, hard disk, software application, and database. The term cloud refers to a network or the internet [6]. It is a technology that uses remote servers on the internet to store, manage, and access data online rather than local drives. The data can be anything such as files, images, documents, audio, video, and more.

Types of Cloud

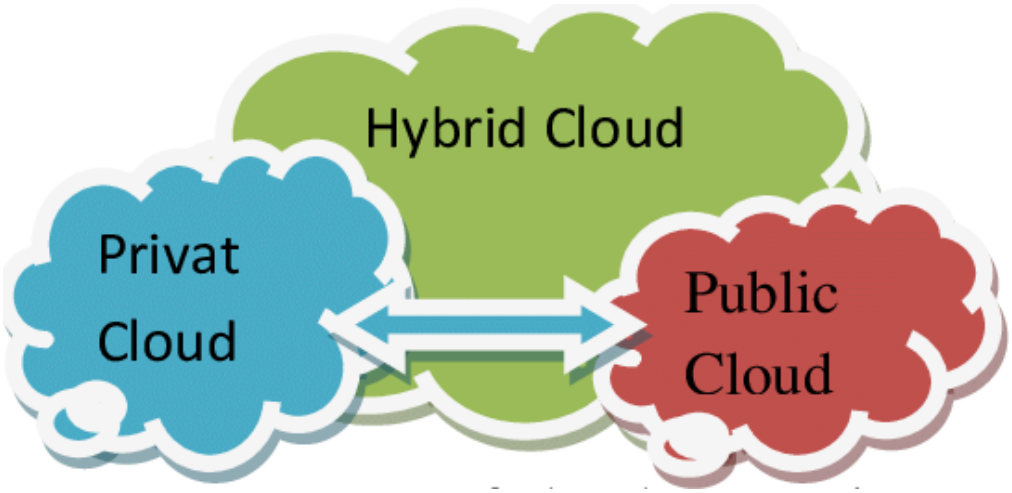


Figure 15: This figure tells us about the types of Cloud.

Cloud Service Models

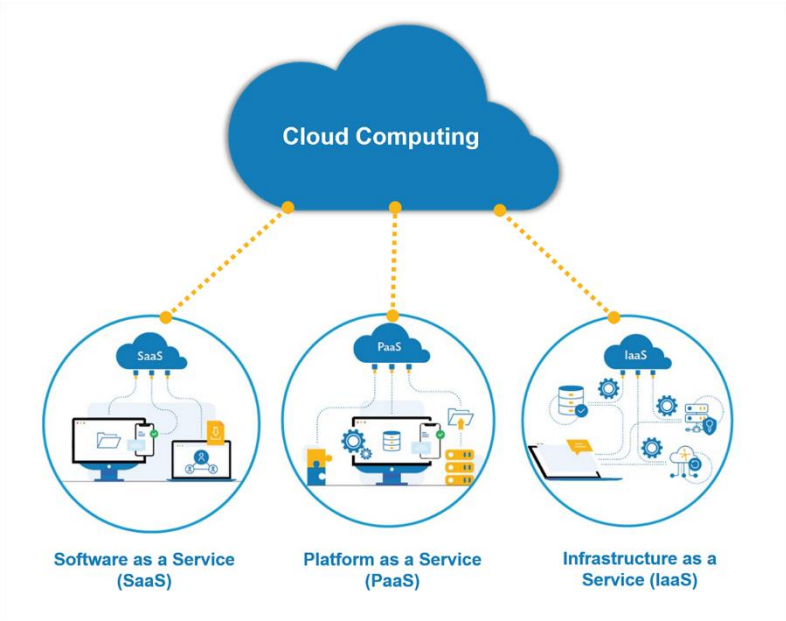


Figure 16: This figure tells us about the Service Models of Cloud.

2.7 Amazon Web Services

AWS (Amazon Web Services) is a **secure cloud service platform** provided by **Amazon**. It offers various services such as database storage, computing power, content delivery, Relational Database, Simple Email, Simple Queue, and other functionality to increase the organization's growth.

Figure 17: This figure tells us about the logo of Amazon web services.

Features of AWS

AWS provides various powerful features for building scalable, cost-effective, enterprise applications. Some important features of AWS are given below-

- AWS is **scalable** because it has an ability to scale the computing resources up or down according to the organization's demand.
- AWS is **cost-effective** as it works on a **pay-as-you-go** pricing model.
- It provides various flexible storage options.
- It offers various **security services** such as infrastructure security, data encryption, monitoring & logging, identity & access control, penetration testing, and DDoS attacks.
- It can efficiently manage and secure Windows workloads.

Figure 18: This figure tells us about the Services of AWS.

2.8 Amazon S3

Amazon Simple Storage Service (Amazon S3) is an object storage service that offers industry-leading scalability, data availability, security, and performance. Customers of all sizes and industries can use Amazon S3 to store and protect any amount of data for a range of use cases, such as data lakes, websites, mobile applications, backup and restore, archive, enterprise applications, IoT devices, and big data analytics. Amazon S3 provides management features so that you can optimize, organize, and configure access to your data to meet your specific business, organizational, and compliance requirements [7].

Figure 19: This figure tells us about the AWS S3.

Bucket A bucket is a container for objects stored in Amazon S3. You can store any number of objects in a bucket and can have up to 100 buckets in your account. To request an increase, visit the Service Quotas console. When you create a bucket, you enter a bucket name and choose the AWS Region where the bucket will reside. After you create a bucket, you cannot change the name of the bucket or its Region. Bucket names must follow the bucket naming rules. You can also configure a bucket to use S3 Versioning or other storage management features.

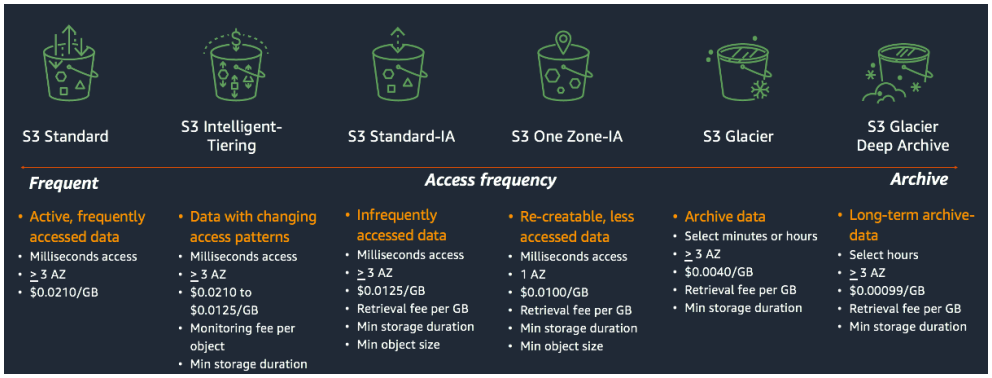


Figure 20: This figure tells us about the Storages Classes of S3.

2.9 Amazon EC2

Amazon Elastic Compute Cloud (Amazon EC2) provides scalable computing capacity in the Amazon Web Services (AWS) Cloud. Using Amazon EC2 eliminates your need to invest in hardware up front, so you can develop and deploy applications faster. You can use Amazon EC2 to launch as many or as few virtual servers as you need, configure security and networking, and manage storage. Amazon EC2 enables you to scale up or down to handle changes in requirements or spikes in popularity, reducing your need to forecast traffic [8].

Figure 21: This figure tells us about the AWS EC2.

2.10 Load Balancer

Load balancing is the method of distributing network traffic equally across a pool of resources that support an application. Modern applications must process millions of users simultaneously and return the correct text, videos, images, and other data to each user in a fast and reliable manner [9].

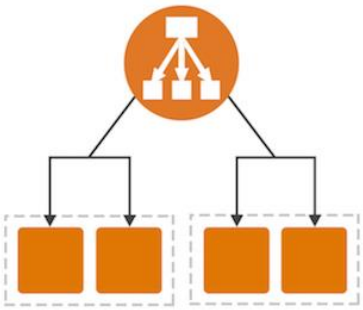


Figure 22: This figure tells us about the Load Balancer.

2.11 VPC & VPC Peering

With **Amazon Virtual Private Cloud (Amazon VPC),** you can launch AWS resources in a logically isolated virtual network that you've defined. This virtual network closely resembles a traditional network that you'd operate in your own data center, with the benefits of using the scalable infrastructure of AWS.

The following diagram shows an example VPC. The VPC has one subnet in each of the Availability Zones in the Region, EC2 instances in each subnet, and an internet gateway to allow communication between the resources in your VPC and the internet [10].

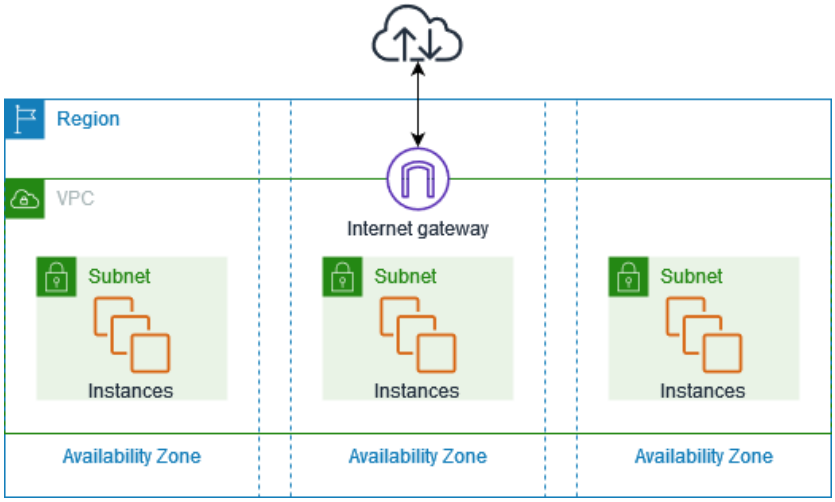


Figure 23: This figure tells us about the VPC Service in AWS.

VPC Peering A VPC peering connection is a networking connection between two VPCs that enables you to route traffic between them using private IPv4 addresses or IPv6 addresses. Instances in either VPC can communicate with each other as if they are within the same network [11]. You can create a VPC peering connection between your own VPCs, or with a VPC in another AWS account. The VPCs can be in different Regions (also known as an inter-Region VPC peering connection).

Kinds of Peering

1. Same account but different regions
2. Different account and different region
3. Different account but same regions

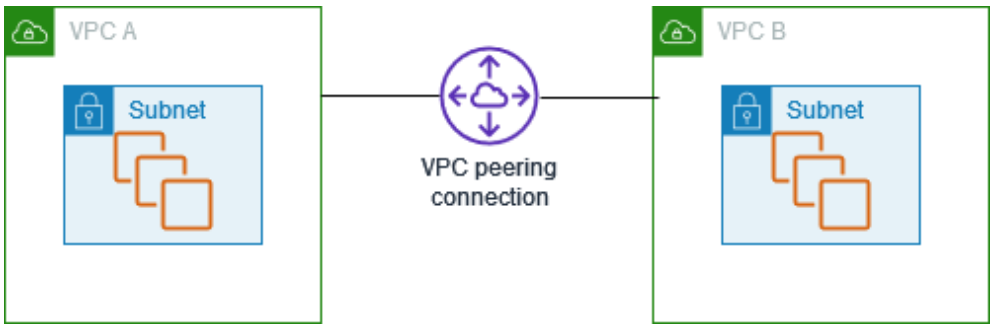


Figure 24: This figure tells us about the VPC Peering Service in AWS.

2.12 Auto Scaling Group

An **Auto Scaling group** contains a collection of EC2 instances that are treated as a logical grouping for the purposes of automatic scaling and management. An Auto Scaling group also lets you use Amazon EC2 Auto Scaling features such as health check replacements and scaling policies. Both maintaining the number of instances in an Auto Scaling group and automatic scaling are the core functionality of the Amazon EC2 Auto Scaling service [12].

The size of an Auto Scaling group depends on the number of instances that you set as the desired capacity. You can adjust its size to meet demand, either manually or by using automatic scaling.

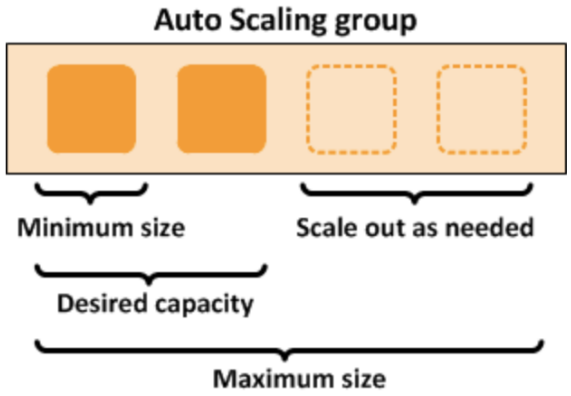


Figure 25: This figure tells us about the Auto Scaling Group Service in AWS.

2.13 Amazon RDS

Amazon Relational Database Service (Amazon RDS) is a web service that makes it easier to set up, operate, and scale a relational database in the AWS Cloud. It provides cost-efficient, resizable capacity for an industry-standard relational database and manages common database administration tasks [13].

Figure 26: This figure tells us about the RDS in AWS.

2.14 Microsoft Azure

Microsoft Azure is also known as **Windows Azure**. It supports various operating systems, databases, programming languages, frameworks that allow IT professionals to easily build, deploy, and manage applications through a worldwide network. It also allows users to create different groups for related utilities.

Figure 27: This figure tells us about the logo of Microsoft Azure.

Features of Microsoft Azure

- Microsoft Azure provides **scalable**, **flexible**, and **cost-effective**
- It allows developers to quickly manage applications and websites.
- It managed each resource individually.
- Its IaaS infrastructure allows us to launch a general-purpose virtual machine in different platforms such as Windows and Linux.
- It offers a **Content Delivery System (CDS)** for delivering the Images, videos, audios, and applications.

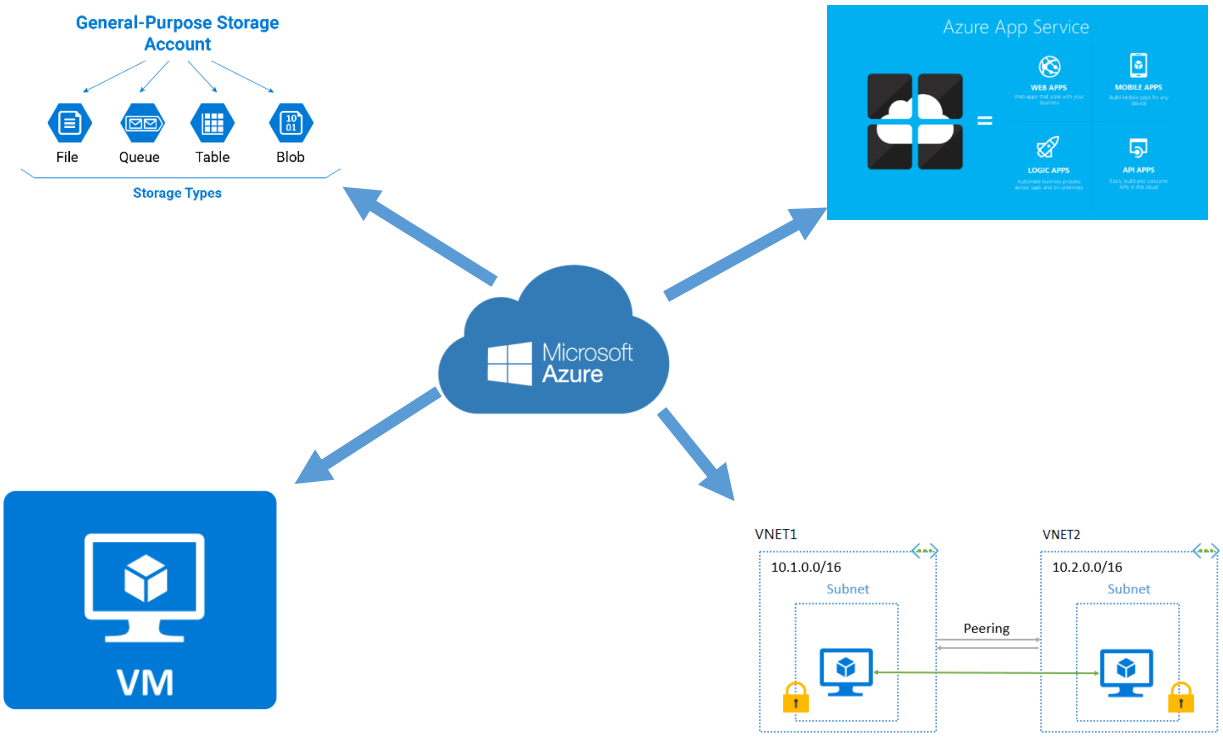


Figure 28: This figure tells us about the Services of Microsoft Azure.

PROGRAMMING Session

2.15 Python

Python is an interpreted, object-oriented, high-level programming language with dynamic semantics developed by Guido van Rossum. Python is used for server-side web development, software development, mathematics, and system scripting, and is popular for Rapid Application Development and as a scripting or glue language to tie existing components because of its high-level, built-in data structures, dynamic typing, and dynamic binding.

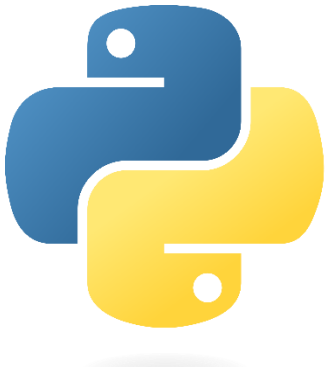


Figure 29: This figure tells us about the logo of Python.

2.16 Bash

➔ **Bash** is CLI (Command Line Interpreter) for interacting with the computer from the command line.
➔ Can be used to write shell scripts.
➔ Location is /bin/bash.
➔ Bash prompt:
- $ for regular user.
- # for root user.
➔ which $SHELL (output:: /bin/bash) Here, which command tells the path of the shell
➔ Write programs or commands that can be run on the system from the command line.
➔ Extension: .sh
➔ Bash script first line is: #! /bin/bash (shebang)

Figure 30: This figure tells us about the BASH.

2.17 Amazon Lambda Function

AWS Lambda is a serverless compute service that runs your code in response to events and automatically manages the underlying compute resources for you. These events may include changes in state or an update, such as a user placing an item in a shopping cart on an ecommerce website. You can use AWS Lambda to extend other AWS services with custom logic, or create your own backend services that operate at AWS scale, performance, and security. AWS Lambda automatically runs code in response to multiple events, such as HTTP requests via Amazon API Gateway, modifications to objects in Amazon Simple Storage Service (Amazon S3) buckets, table updates in Amazon DynamoDB, and state transitions in AWS Step Functions.

Lambda runs your code on high availability compute infrastructure and performs all the administration of your compute resources. This includes server and operating system maintenance, capacity provisioning and automatic scaling, code and security patch deployment, and code monitoring and logging. All you need to do is supply the code.

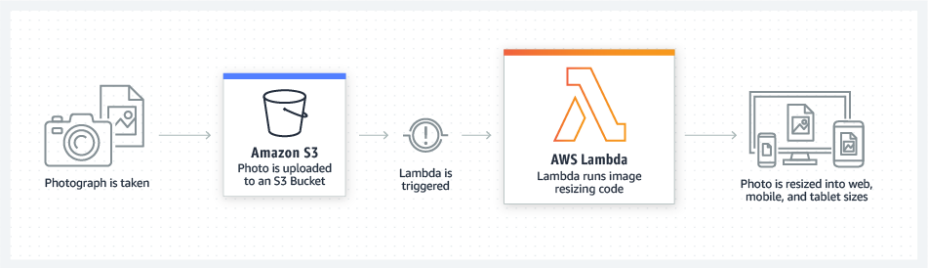


Figure 31: This figure tells us about the AWS Lambda how its work.

2.18 Azure Function

Azure Functions is a serverless solution that allows you to write less code, maintain less infrastructure, and save on costs. Instead of worrying about deploying and maintaining servers, the cloud infrastructure provides all the up-to-date resources needed to keep your applications running.

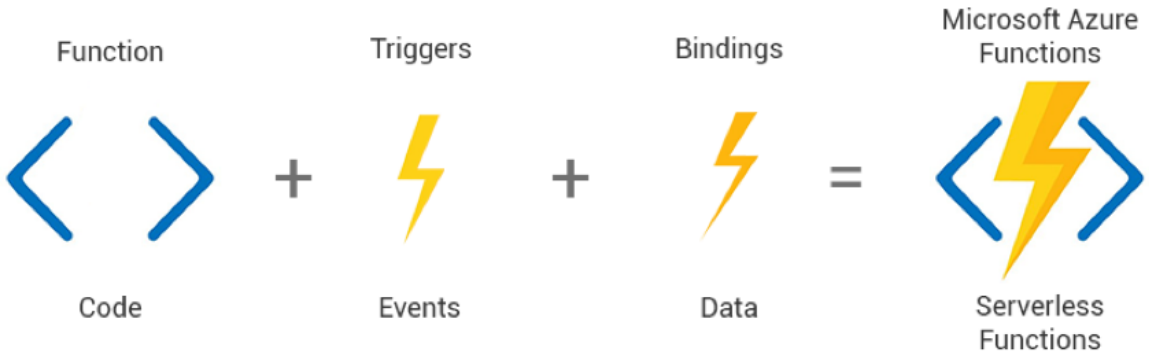


Figure 32: This figure tells us about the Azure Function how its work.

Chapter 3: Introduction to Project
3.1 Introduction to Project

The Automated Infrastructure Deployment with Terraform and GitHub Actions project brings together the power of Terraform and GitHub Actions to revolutionize the process of provisioning and managing infrastructure. In today's rapidly evolving technological landscape, organizations require efficient and scalable solutions for deploying cloud resources. By leveraging infrastructure as code principles and automating the deployment pipeline, this project offers a comprehensive solution to meet those needs.

Infrastructure as code, as exemplified by Terraform, allows for the definition and management of infrastructure resources using code. This approach offers numerous benefits, including version control, reproducibility, and easy collaboration. Terraform enables users to describe their desired infrastructure state using a declarative syntax, making it easier to maintain and modify infrastructure configurations over time.

GitHub Actions, a robust automation platform, serves as the foundation for the deployment pipeline in this project. By leveraging GitHub Actions, users can define workflows that automate tasks such as infrastructure testing, validation, and deployment. With its flexibility and integration capabilities, GitHub Actions becomes the catalyst for continuous integration and deployment, ensuring that infrastructure changes are tested and deployed consistently and reliably.

The project's key objective is to provide an end-to-end solution for automated infrastructure deployment. By combining Terraform's infrastructure provisioning capabilities with GitHub Actions' automation workflows, users can achieve a streamlined and efficient deployment process. The project incorporates features such as automated testing and validation, environment-specific workflows, secret management, and monitoring to enhance the deployment pipeline and improve overall infrastructure quality.

Automated Infrastructure Deployment with Terraform and GitHub Actions offers a wide range of benefits. By automating the deployment process, manual effort is significantly reduced, allowing developers and system administrators to focus on higher-level tasks. The use of infrastructure as code ensures reproducibility and scalability, allowing for easy replication of infrastructure across different environments. Collaborative features provided by GitHub, such as version control and code review, enable effective team collaboration and facilitate knowledge sharing.

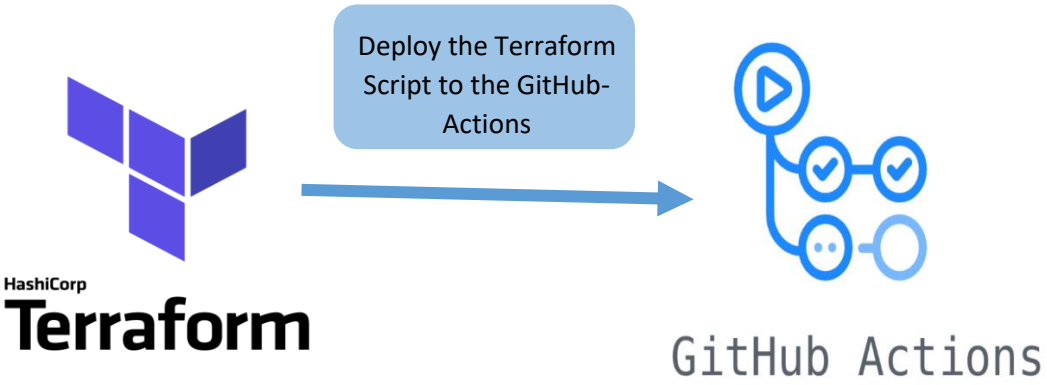


Figure 33: This figure tells us about the terraform deploy to GitHub – Actions.

3.2 Project Category

DevOps refers to the practice of combining development (Dev) and operations (Ops) to streamline and automate the software development lifecycle. It focuses on collaboration, communication, and integration between development teams and operations teams to deliver applications more efficiently and reliably.

In this project, automation is achieved by leveraging Terraform and GitHub Actions. Terraform allows for the provisioning and management of infrastructure resources as code, promoting infrastructure as code practices. GitHub Actions, on the other hand, automates workflows and processes, including infrastructure testing, validation, and deployment.

By utilizing Terraform and GitHub Actions together, the project embodies DevOps principles by automating the infrastructure deployment process, improving collaboration, reducing manual effort, and ensuring consistent and reliable infrastructure deployments.

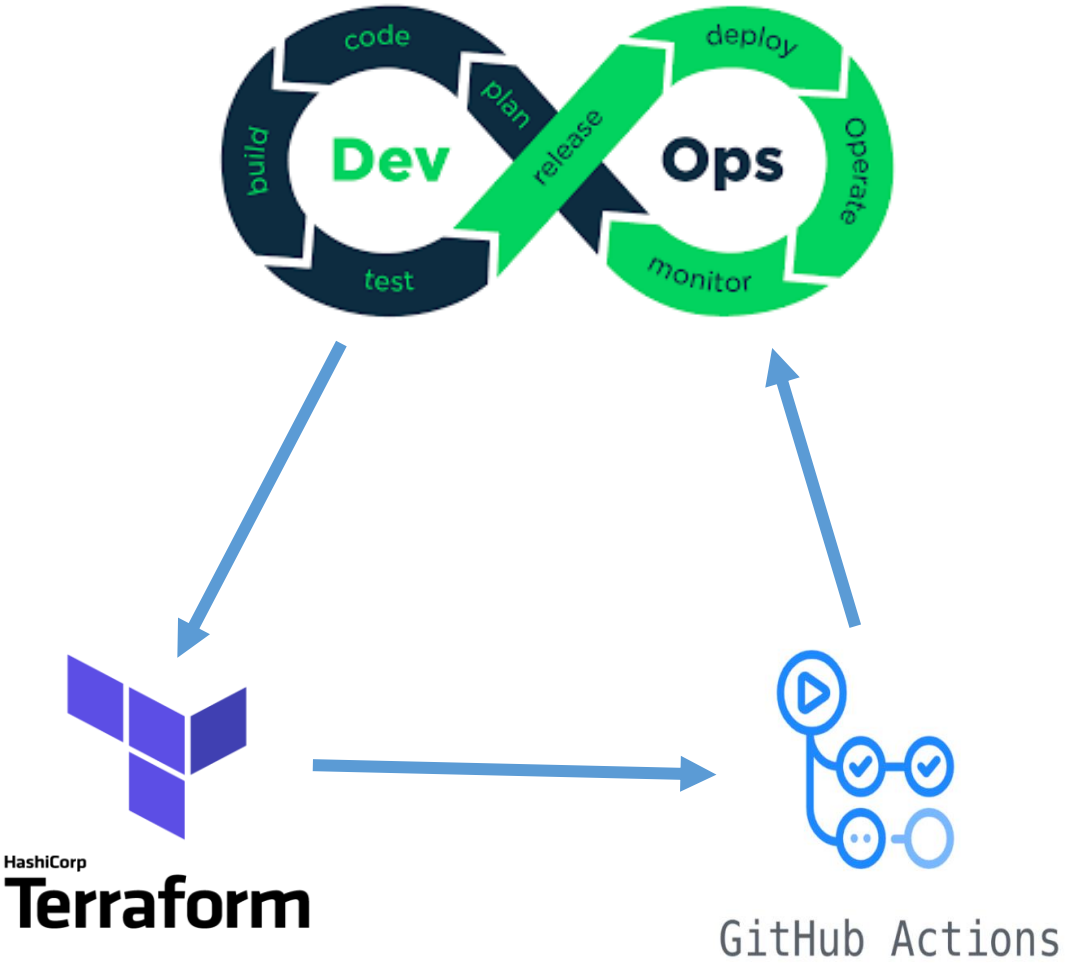


Figure 34: This figure tells us about the in which Categpry Project undergoes.

3.3 Objectives

The objectives of the project "Automated Infrastructure Deployment with Terraform and GitHub Actions" are as follows:

1. Streamline Infrastructure Deployment: The primary objective is to create an automated and streamlined process for deploying infrastructure resources. By leveraging Terraform and GitHub Actions, the project aims to eliminate manual and error-prone processes, enabling faster and more reliable infrastructure deployments.

2. Infrastructure as Code: The project aims to promote the adoption of infrastructure as code principles. By using Terraform, infrastructure configurations are defined and managed as code, allowing for version control, reproducibility, and easier collaboration among team members.

3. Continuous Integration and Deployment: The project seeks to establish continuous integration and deployment practices for infrastructure changes. With GitHub Actions, infrastructure code can be automatically tested, validated, and deployed based on triggers such as code pushes or pull requests, ensuring that infrastructure changes are thoroughly tested and deployed consistently.

4. Automation of Testing and Validation: The project aims to automate the testing and validation of infrastructure code. This includes running tests, performing static code analysis, and validating the correctness of infrastructure configurations. By automating these processes, the project ensures the reliability and quality of deployed infrastructure.

5. Secure and Scalable Infrastructure Deployment: The project aims to incorporate security and scalability aspects into the infrastructure deployment pipeline. This includes securely managing credentials and API keys using GitHub Secrets, implementing best practices for security and compliance, and allowing for easy scaling of infrastructure resources as the project requirements evolve.

6. Collaboration and Transparency: The project promotes collaboration and transparency among team members. By hosting the infrastructure code on GitHub, it allows for code review, discussions, and feedback. The use of version control enables tracking of changes, providing visibility into infrastructure modifications and facilitating collaboration within the team.

7. Monitoring and Reporting: The project aims to integrate monitoring and reporting capabilities into the infrastructure deployment pipeline. This includes setting up alerts and notifications for critical infrastructure events and generating reports to track the status of deployments. By monitoring the deployed infrastructure, the project ensures its health and performance.

Overall, the objectives of the project are to automate infrastructure deployment, enforce infrastructure as code practices, enable continuous integration and deployment, ensure security and scalability, promote collaboration, and incorporate monitoring and reporting for effective infrastructure management.

3.4 Problem Formulation

Problem: Manual and error-prone infrastructure deployment processes hinder efficiency, reliability, and scalability for organizations. Lack of version control, inadequate testing, and labour-intensive procedures lead to delays, inconsistencies, and potential security vulnerabilities.

Objective: Develop an automated infrastructure deployment pipeline using Terraform and GitHub Actions to address the challenges and improve the overall infrastructure provisioning and management process.

Key Challenges:

1. **Manual Deployment Processes**: Manual infrastructure deployment processes are time-consuming, error-prone, and require significant manual effort. Manually executing tasks such as provisioning resources, configuration management, and tracking changes increase the risk of inconsistencies and human errors.
2. **Lack of Version Control**: Without proper version control mechanisms, managing infrastructure configurations becomes complex and prone to conflicts. Tracking changes, coordinating collaboration, and rolling back to previous configurations become challenging without a centralized version control system.
3. **Limited Testing and Validation**: Traditional infrastructure deployment methods often lack automated testing and validation processes. The absence of automated tests and validations increases the chances of misconfigurations, security vulnerabilities, and performance issues going undetected, leading to potentially unstable or insecure infrastructure.
4. **Inefficient Collaboration**: Inadequate collaboration mechanisms hinder effective teamwork. Siloed processes, lack of visibility into infrastructure changes, and difficulties in reviewing and providing feedback on infrastructure configurations can slow down development cycles and impede collaboration among team members.
5. **Security Risks**: Managing sensitive information, such as credentials and API keys, in an insecure manner can expose the infrastructure to security risks. Ensuring secure storage and access control of these credentials is critical for maintaining the integrity and confidentiality of the infrastructure.
6. **Scalability Challenges**: Scaling infrastructure resources manually can be time-consuming and error prone. Lack of automated scaling mechanisms can result in inefficient resource allocation and hinder the ability to adapt to changing workload demands.

Solution Approach: The project aims to address these challenges by leveraging Terraform and GitHub Actions to automate the infrastructure deployment pipeline. Infrastructure configurations will be defined as code using Terraform, allowing for version control, easy collaboration, and reproducibility. GitHub Actions will enable the automation of testing, validation, and deployment processes, ensuring consistent and reliable infrastructure deployments. By incorporating secure secret management, scalability mechanisms, and fostering collaboration, the project aims to provide an efficient and scalable solution for automated infrastructure deployment.

Success Metrics:

1. **Reduction in Manual Effort:** Measure the decrease in manual effort required for infrastructure deployment tasks, such as provisioning resources, configuration management, and tracking changes.

2. **Improved Deployment Speed**: Assess the speed of infrastructure deployment by comparing the time taken for manual deployments versus automated deployments using the project pipeline.

3. **Enhanced Infrastructure Quality**: Evaluate the quality of deployed infrastructure by measuring the reduction in misconfigurations, security vulnerabilities, and performance issues through automated testing and validation processes.

4. **Increased Collaboration and Transparency:** Gauge the level of collaboration and transparency within the team by tracking the number of code reviews, discussions, and feedback provided on infrastructure changes.

5. **Secure Credential Management**: Ensure the secure management of credentials and API keys by assessing the effectiveness of the implemented secret management mechanisms.

6. **Scalability and Adaptability**: Measure the ability to scale infrastructure resources efficiently in response to changing workload demands, ensuring optimal resource allocation and performance.

By addressing these problem areas and achieving the defined success metrics, the project aims to deliver an automated infrastructure deployment pipeline that improves efficiency, reliability, security, scalability, and collaboration for organizations.

3.5 Existing System

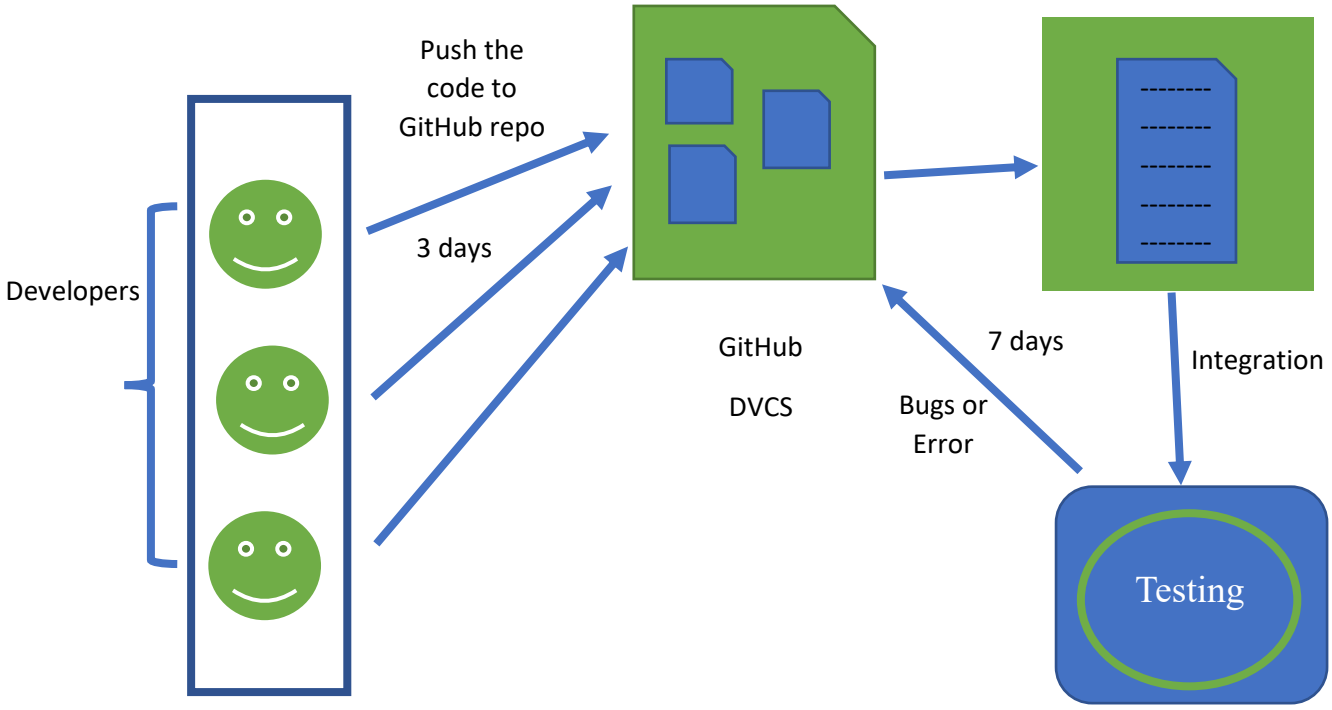


Figure 35: This figure tells us about the Existing System without CI/CD pipeline.

In earlier times, developers used to push their code on GitHub, then the integration of the code would mean adding all the codes, then this integrated code was sent to the testing team, if the tester had any error or bug, then the testing teams would fix that Used to send notifications to GitHub, due to this the developers used to face a lot of trouble in finding the error, till then the developer was busy in some other work.

3.6 Proposed System

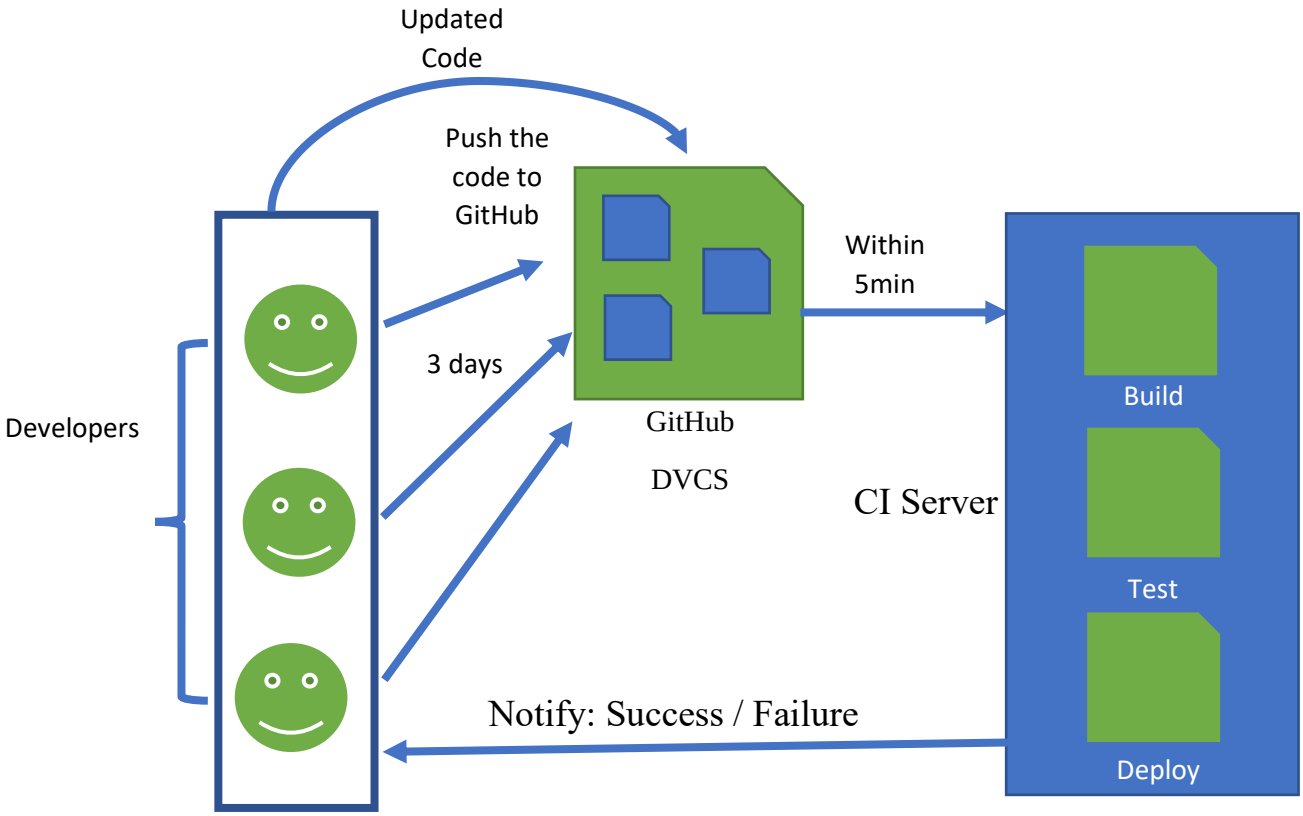


Figure 36: This figure tells us about the Proposed System with CI/CD pipeline.

In the present times, instead of sending the entire code, the developers used to push small codes on GitHub, then send the code to the integration server where the code is built and then tested, then finally deployed. Means it is not deployed internally in production environment, this whole process is done within 5 minutes, then it notifies Rahul Develops whether it is successful or failure. updated code again push.

3.7 Unique Features of the System

A CI/CD (Continuous Integration/Continuous Deployment) pipeline incorporates several features to automate and streamline the software development and deployment processes. The key features of a CI/CD pipeline are as follows:

1. **Continuous Integration**: This feature involves automatically building and integrating code changes from multiple developers into a shared repository. It ensures that the changes are merged and tested in a central codebase, promoting collaboration and early detection of integration issues.
2. **Automated Builds**: The CI/CD pipeline automatically builds the software application or service using build tools and scripts. It ensures that the code compiles successfully and generates the required artifacts, such as executable files or packages, in a consistent and reproducible manner.
3. **Code Quality Checks**: The pipeline incorporates automated code quality checks, including static code analysis, unit tests, and code formatting. These checks ensure that the code adheres to coding standards, best practices, and quality guidelines. Any issues or violations are reported, allowing developers to address them early in the development process.
4. **Automated Testing**: The CI/CD pipeline includes automated testing processes, such as unit tests, integration tests, and functional tests. These tests verify the correctness, functionality, and performance of the application. By automating testing, it helps catch bugs and regressions early, ensuring the stability and reliability of the software.
5. **Artifact Management**: The pipeline manages and stores the generated artifacts, such as executable files, packages, or containers, in a centralized repository. This ensures version control, traceability, and easy accessibility of the build artifacts for deployment purposes.
6. **Deployment Automation**: The CI/CD pipeline automates the deployment of the application or service to various environments, such as development, staging, or production. It uses deployment scripts, infrastructure as code tools, or containerization technologies to ensure consistent and reliable deployments.
7. **Configuration Management**: The pipeline manages the configuration of the application or service across different environments. It ensures that the configuration settings are properly applied during deployments and can be easily managed and modified without manual intervention.
8. **Release Management**: The pipeline facilitates the management and tracking of software releases. It provides versioning, release notes generation, and release tagging mechanisms to keep track of changes and enable easy rollbacks if needed.
9. **Continuous Monitoring**: The pipeline integrates monitoring and alerting mechanisms to track the health, performance, and availability of the deployed application or service. It provides real-time insights into the behaviour of the application, enabling prompt actions in case of issues or anomalies.
10. **Feedback and Collaboration**: The pipeline promotes collaboration and feedback among team members. It notifies developers of build or test failures, provides visibility into the pipeline's status, and facilitates code reviews and discussions. This fosters effective teamwork and continuous improvement.

Chapter 4: Requirement Analysis

4.1 SDLC Model

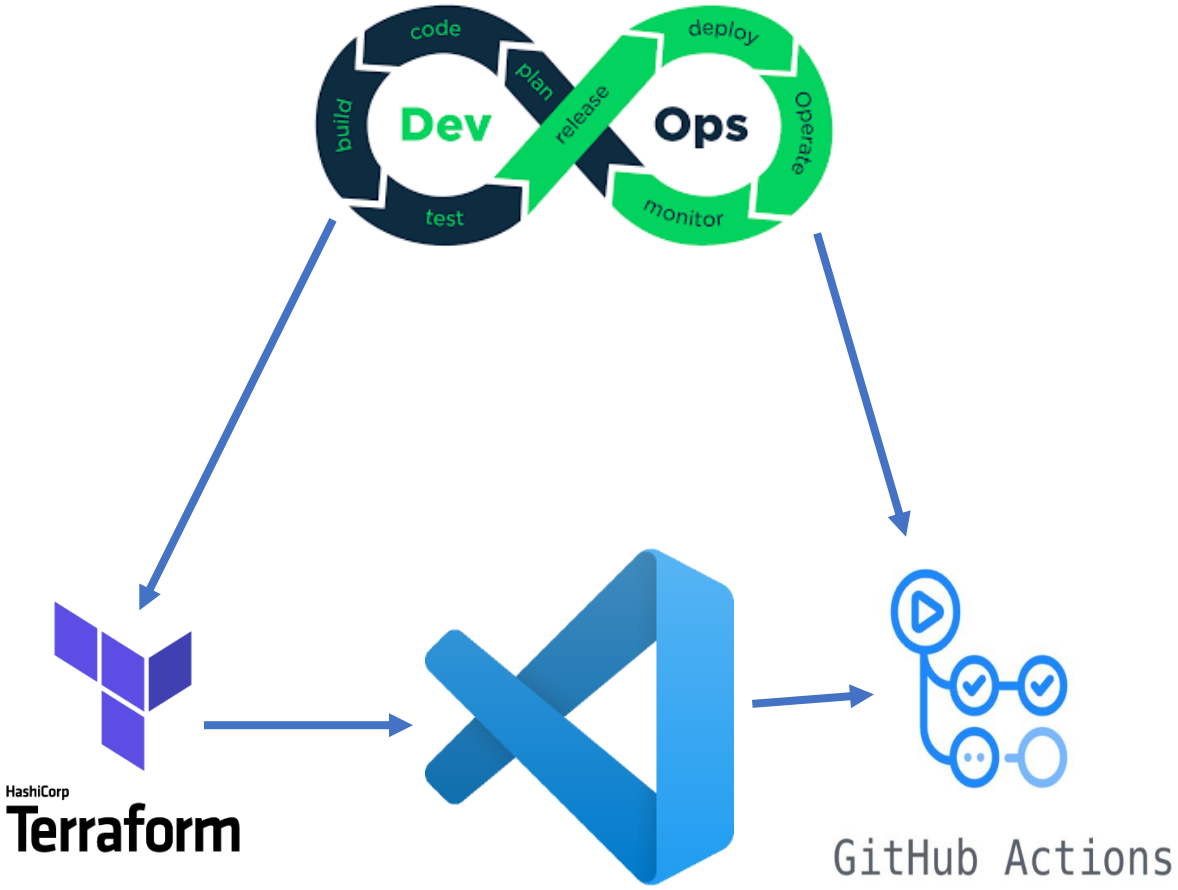


Figure 37: This figure tells us about the SDLC Model of this Project.

In which SDLC model we use DevOps, the terraform is tool for create the infrastructure and write script in Visual Studio code with .tf extensions for testing, I have write the script to make S3 bucket in AWS and the whole script are saved and now next thing open the Git-Hub profile and this GitHub is a tool of CI / CD pipeline in which run the script in workflow as action for creating the resource (S3 bucket) and another action for destroy resource (S3 bucket) by automatically.

Chapter 5: System Design

5.1 Product Functions

1. **Infrastructure Provisioning**: The product allows users to define infrastructure resources such as servers, networks, databases, storage, and other components as code using Terraform. It provides the functionality to specify the desired state of the infrastructure in a declarative manner.
2. **Infrastructure Configuration**: The product enables users to configure the desired settings and parameters of the infrastructure resources within the Terraform code. This includes specifying network configurations, security rules, access control settings, scaling options, and other resource-specific configurations.
3. **Infrastructure Orchestration**: The product provides the capability to orchestrate the deployment and management of the infrastructure resources defined in Terraform. It ensures the correct sequence of resource creation, modification, and deletion, taking dependencies and interrelationships into account.
4. **Continuous Integration:** The product integrates with GitHub Actions to enable continuous integration processes. It allows developers to trigger the execution of automated build and test workflows when changes are pushed to the repository. This ensures that the infrastructure code is validated and tested in an automated and repeatable manner.
5. **Continuous Deployment:** The product facilitates continuous deployment practices by automating the deployment of infrastructure changes. It enables users to define deployment workflows using GitHub Actions, ensuring that infrastructure updates are automatically applied to the target environments upon successful testing and validation.
6. **Version Control and Collaboration**: The product integrates with Git and GitHub, providing version control capabilities for the infrastructure code. It allows multiple team members to collaborate on infrastructure development, review changes, and track modifications over time. This function enables efficient teamwork, code review processes, and facilitates versioning and rollbacks if needed.
7. **Testing and Validation**: The product supports automated testing and validation of the infrastructure code and configurations. It allows users to define and execute various types of tests, including unit tests, integration tests, and validation checks, to ensure the correctness and reliability of the infrastructure.
8. **Deployment Monitoring and Reporting:** The product provides monitoring and reporting functionalities to track the status and health of the deployed infrastructure. It may integrate with monitoring tools or services to collect metrics, generate alerts, and generate reports that provide insights into the performance and availability of the infrastructure resources.

Overall, the product aims to simplify and automate the process of infrastructure provisioning, configuration, and deployment. It enables users to define infrastructure as code, utilize continuous integration and deployment practices, collaborate effectively, and ensure the reliability and scalability of their infrastructure deployments.

5.2 Dataflow / Workflow

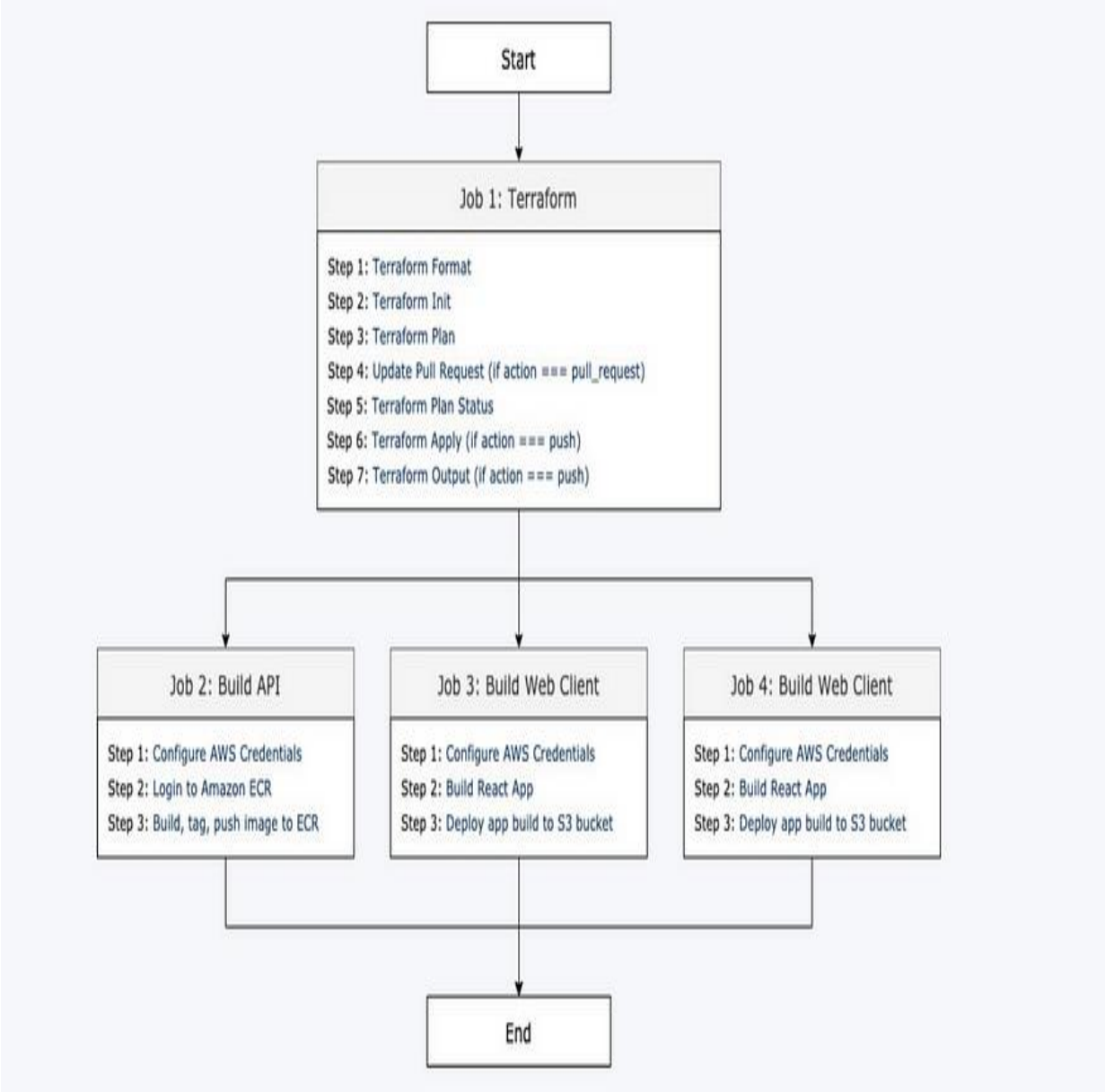
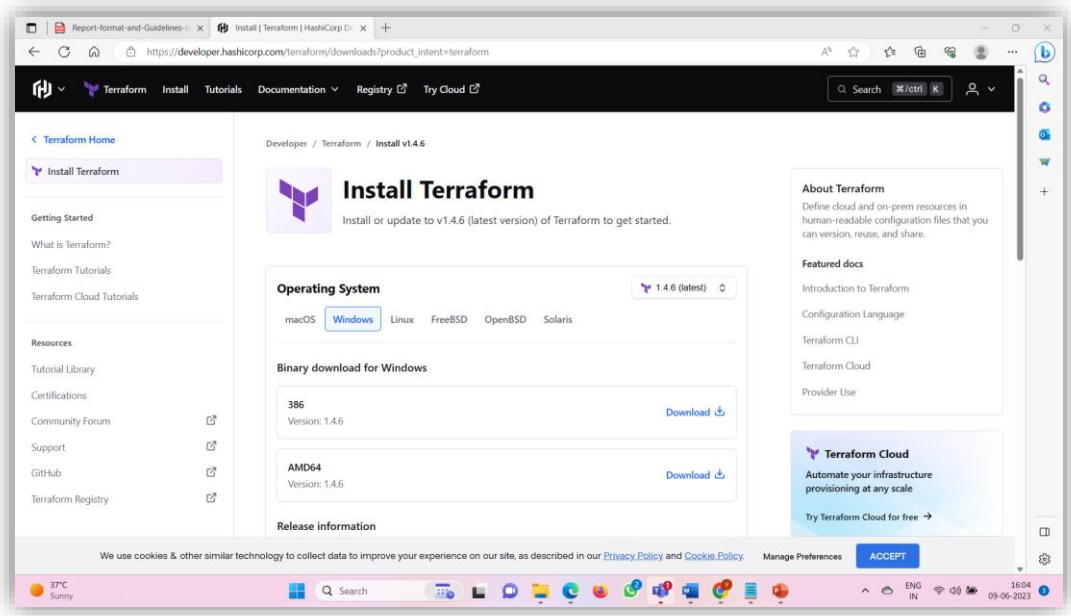


Figure 38: This figure tells us about the Workflow / Dataflow for this project.

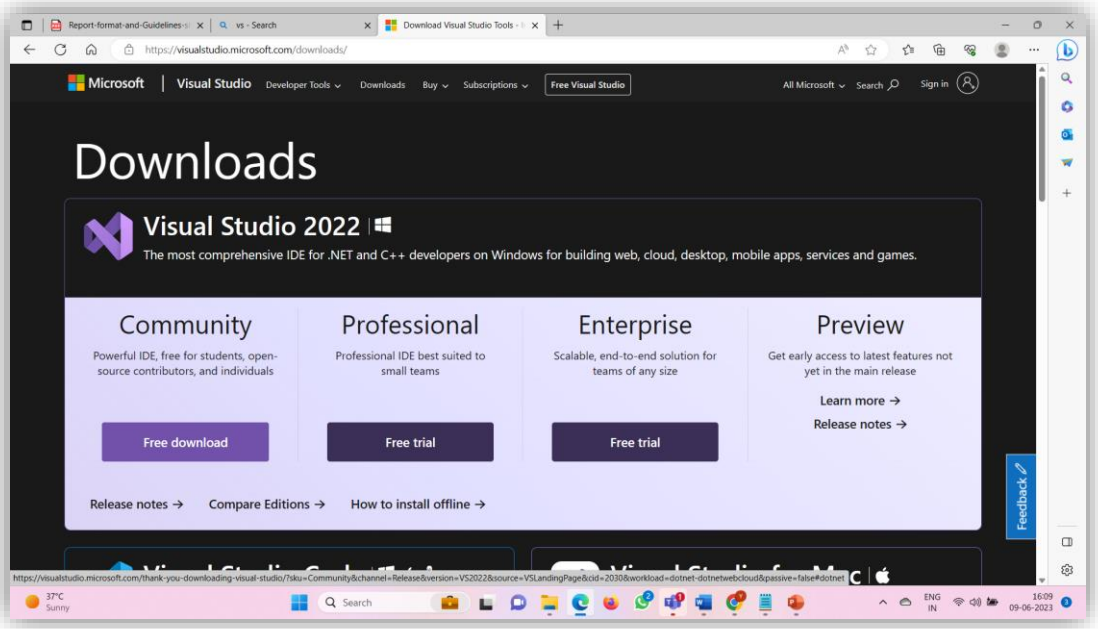
Chapter 6: Development, Implementation and Testing

6.1 Introduction to Languages, IDE's, Tools and Technologies used for Implementation

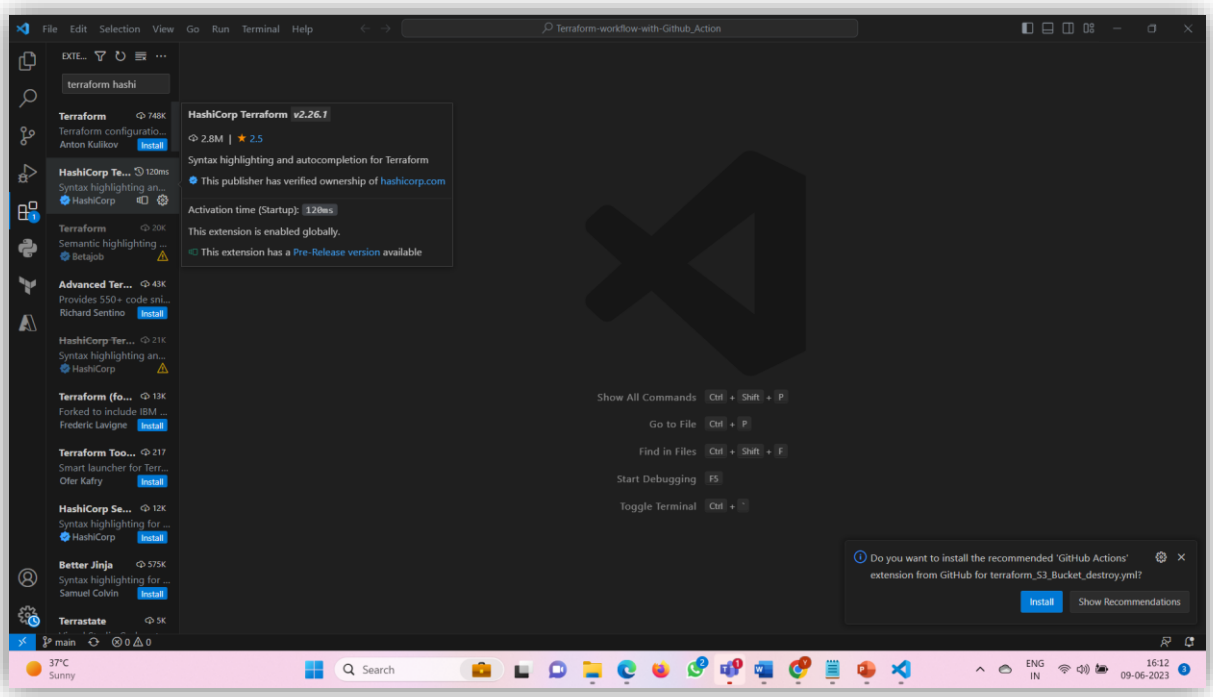
1. Firstly, I was installed the terraform file for this project where I can write the terraform script for that simply search google type terraform and you download the file depend upon your Operating system like that.



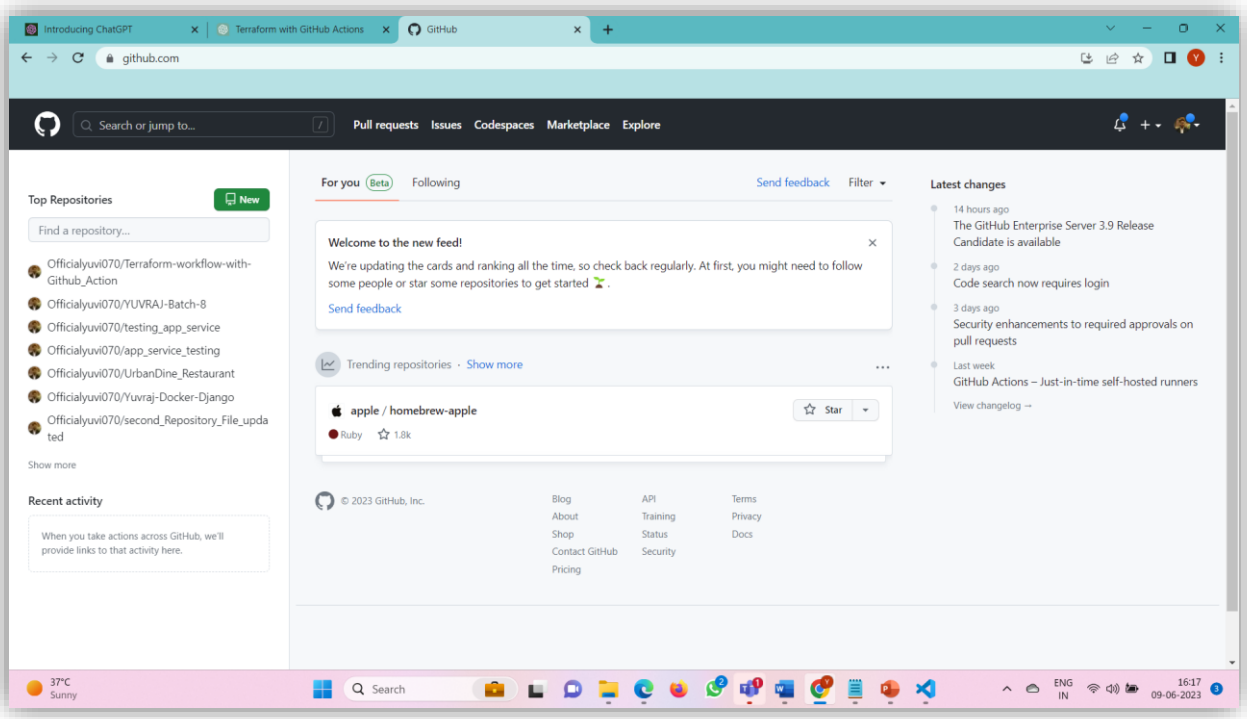
2. Now we need to install the visual Studio for this project according to your operating system like that.



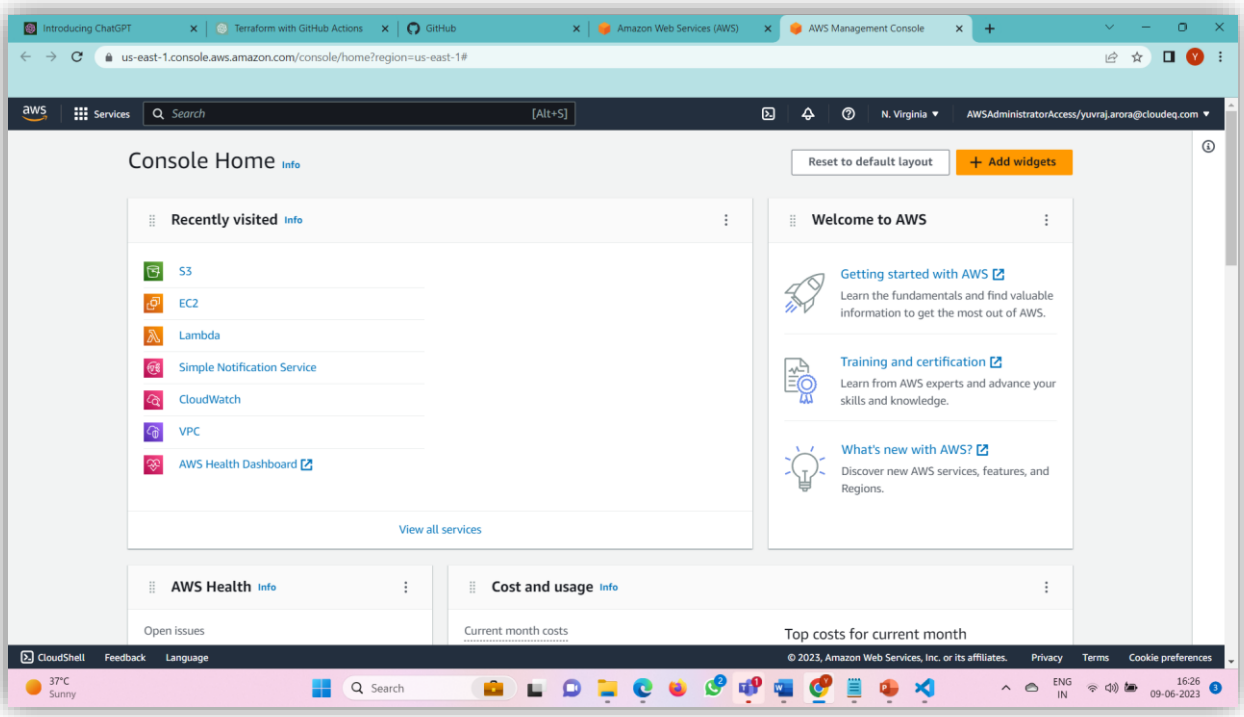
3. Now Download the extension for terraform for this download extension terraform by Hasicorp like that.



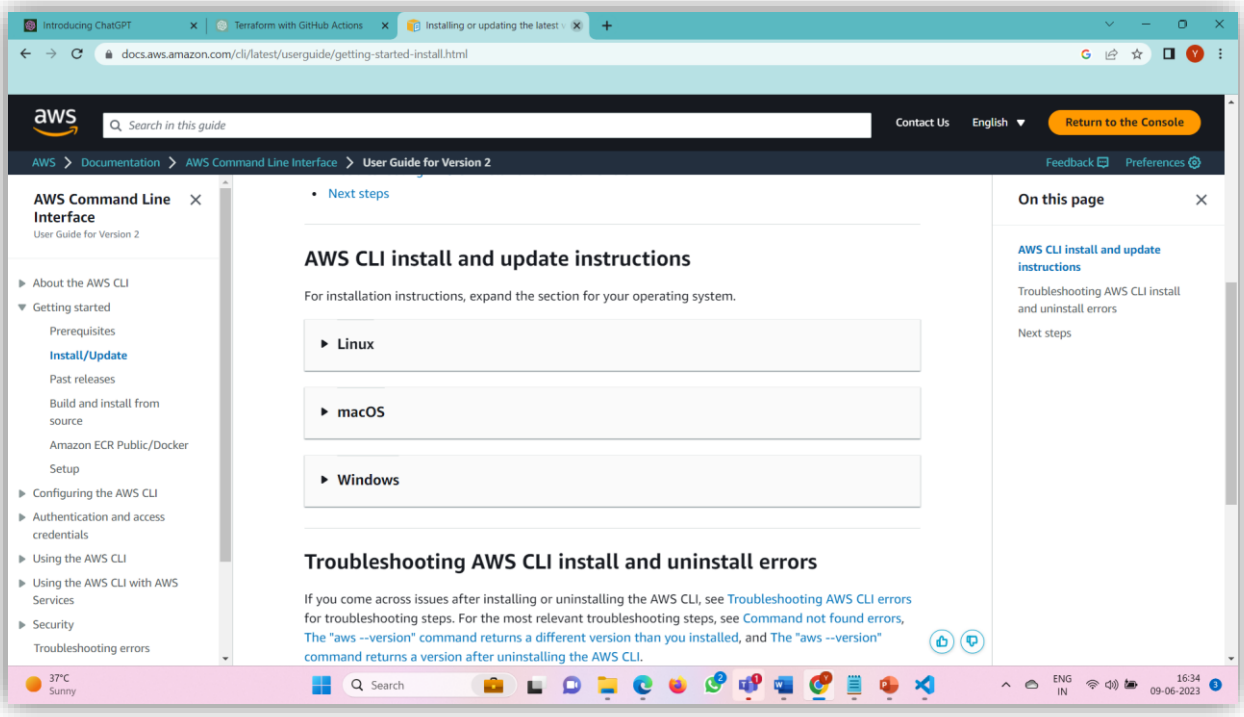
4. Now Create the GitHub Profile that's very easy enter the professional mail id or general mail id we can say enter then password you account has been created.



5. In which I am creating the S3 bucket and Destroy / delete the S3 bucket automatically for this requirement I need AWS account where I can check bucket created or not.



6. Download the AWS CLI (command Line Interface) file in desktop according to operating system like that.



6.2 Coding standards of Language used

For the Terraform write the script in a HCL (**Hashicorp Configuration Language**) in a VS Code. HCL is a JSON compatible language that adds features to help you use the Terraform tool to its highest potential. These features make HCL a powerful configuration language and address some of JSON's shortcomings. For example

```
provider “aws” {
   region = “us-west-1”
}


resource “aws_instance” “myec2” {
   ami = “ami-12345qwert”
   instance_type = “t2.micro”
}
```

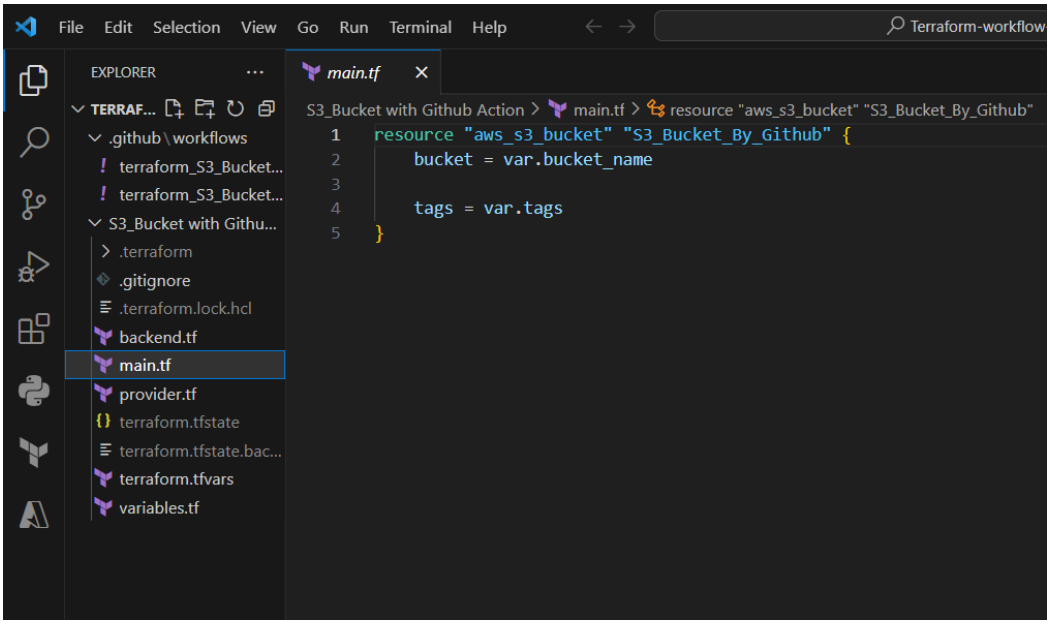
For the GitHub – Action GitHub uses **YAML** syntax to define the workflow. Each workflow is stored as a separate YAML file in your code repository, in a directory named .github/workflows . You can create an example workflow in your repository that automatically triggers a series of commands whenever code is pushed. For example

```
jobs:
  first_demo_job:
    name: The first demo job
    steps:
      - name: Show the demo running
        env:
          VAR1: This is
          VAR2: A Demo of
          VAR3: GitHub Actions
          VAR4: Workflow jobs
        run: |
          echo $VAR1 $VAR2 $VAR3 $VAR4.
<span style="font-weight: 400;">    </span>
```

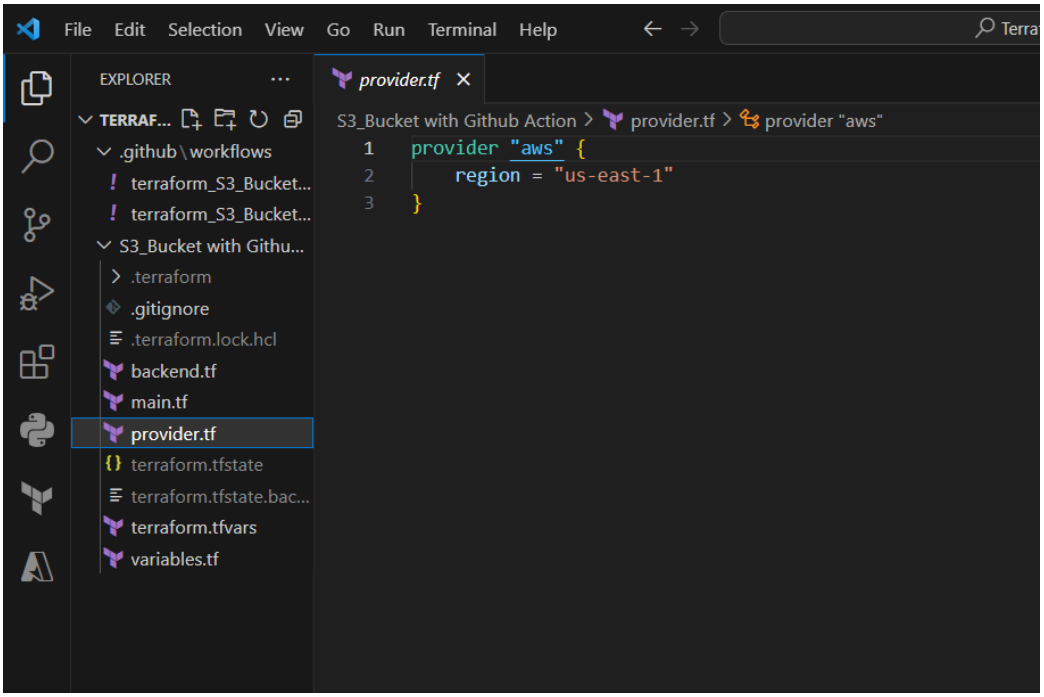
Chapter 7. Results and Discussions

7.1 Snapshots of system with brief detail of each

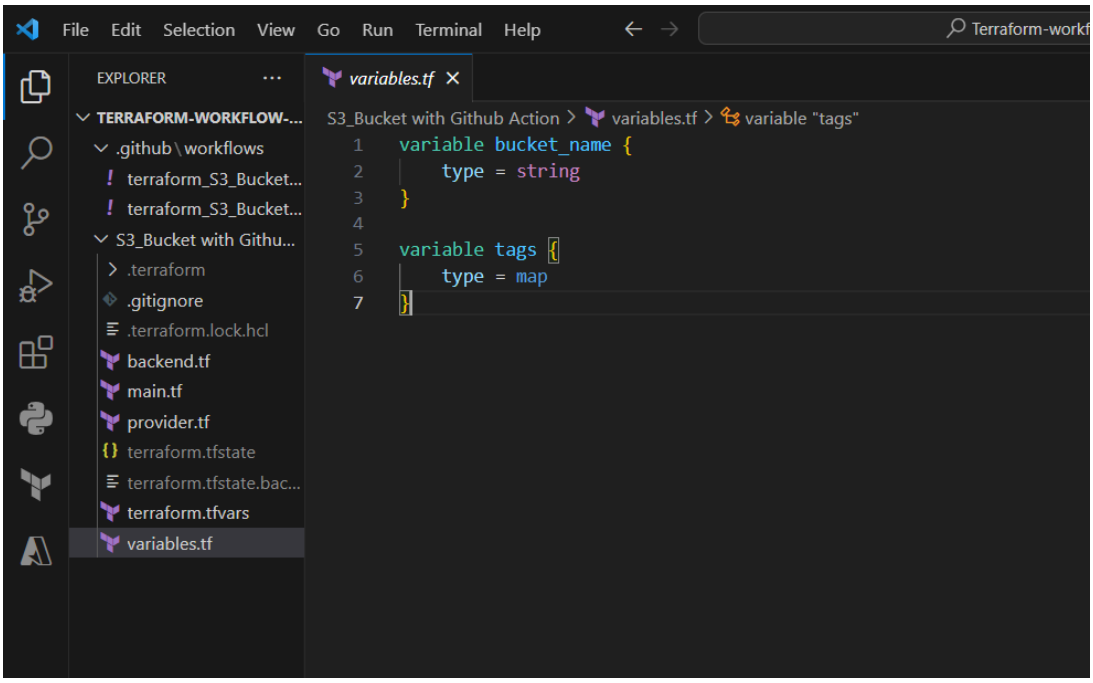
1. Firstly, Open the VS code and write the script for S3 Bucket for that firstly make the mai.tf file in which write main resource which I am creating like that.



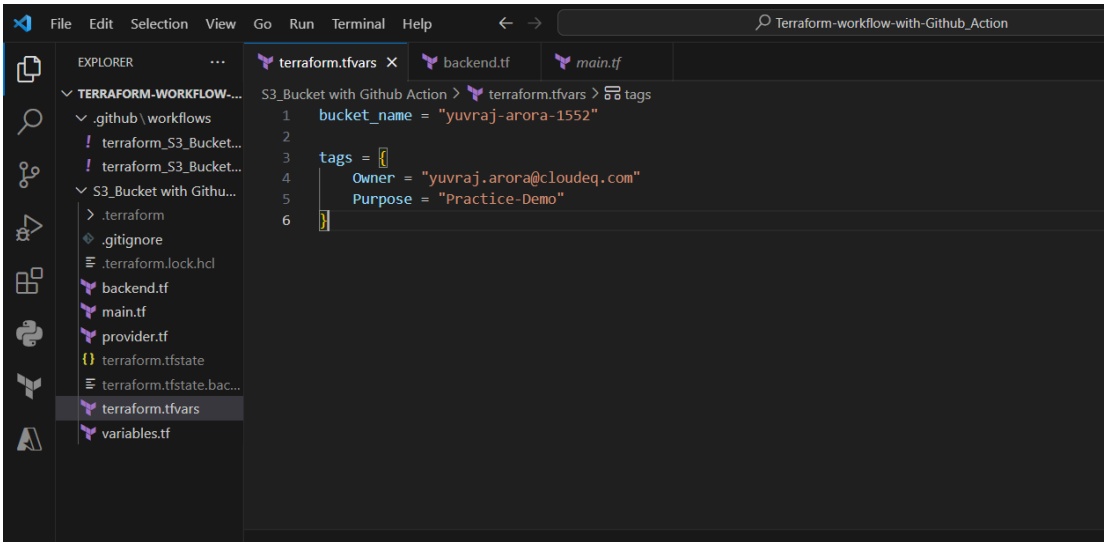
2. Now make the provider.tf file in which describe the location / region of the resource like that.



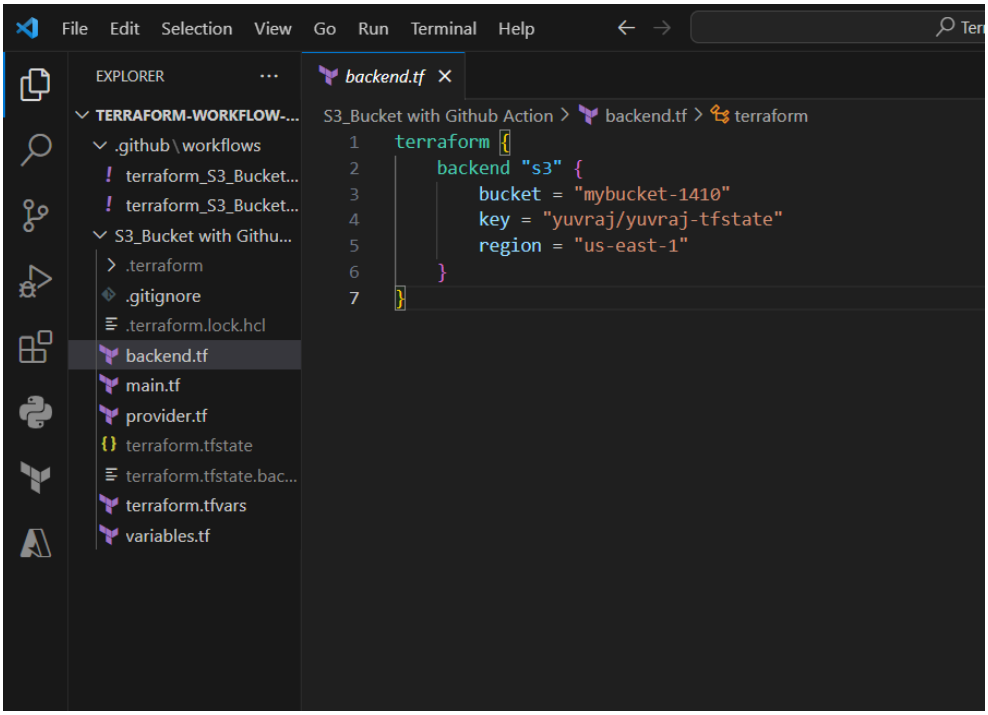
3. Now make variables.tf files in which describes the variables which I am using in that resource like that.



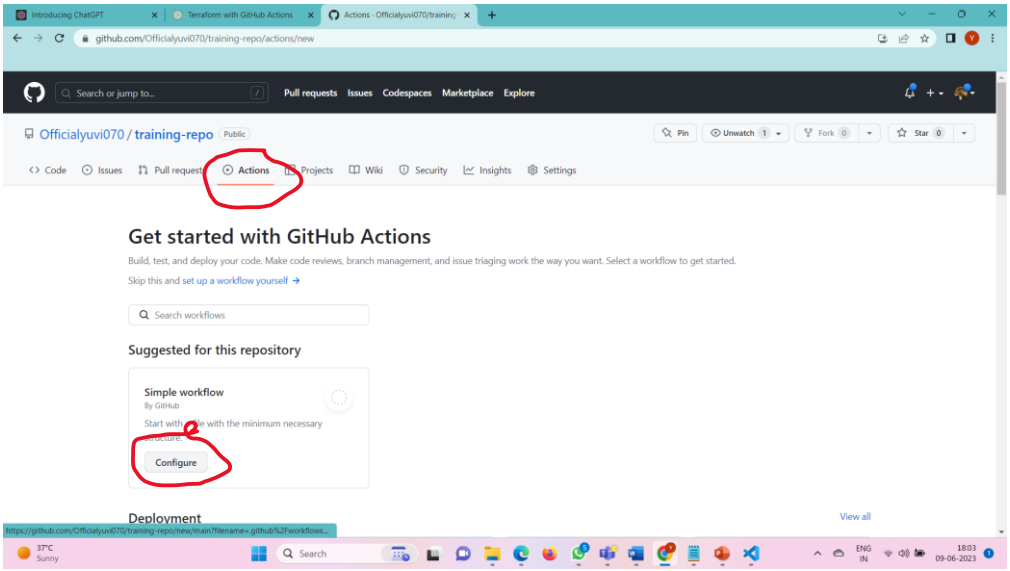
4. Now make another file terraform.tfvars in which describe the values of variables like that.



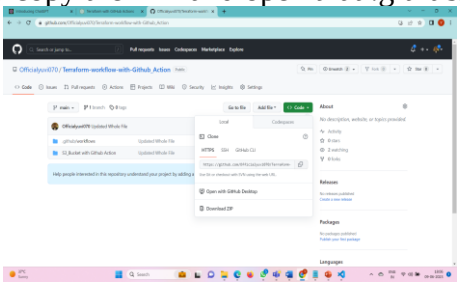
5. Now make another file backend.tf this file is used for the backup.



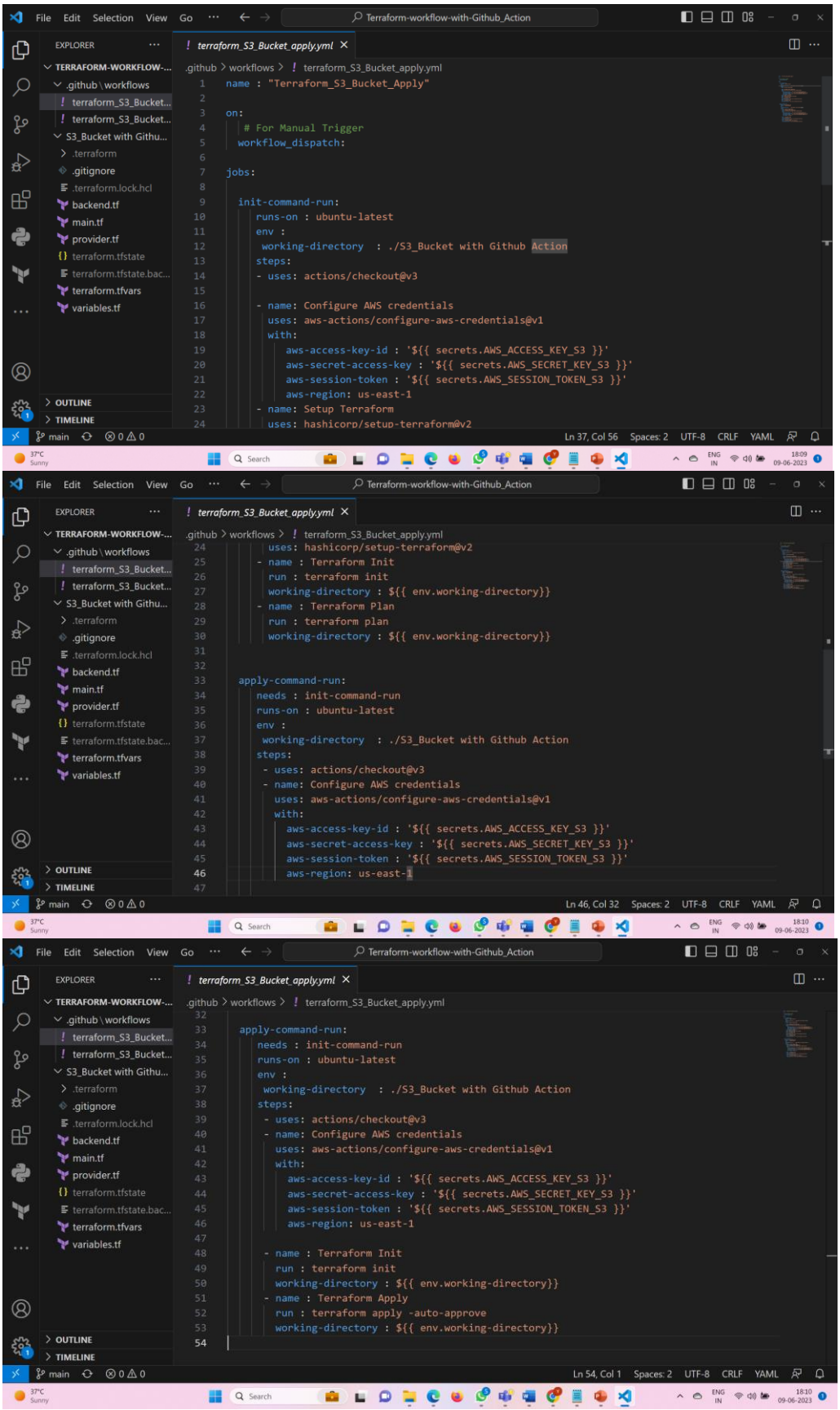
6. Now Open the GitHub Profile and create the **new repository** and click on **Action** and next click on **Simple Workflow**.



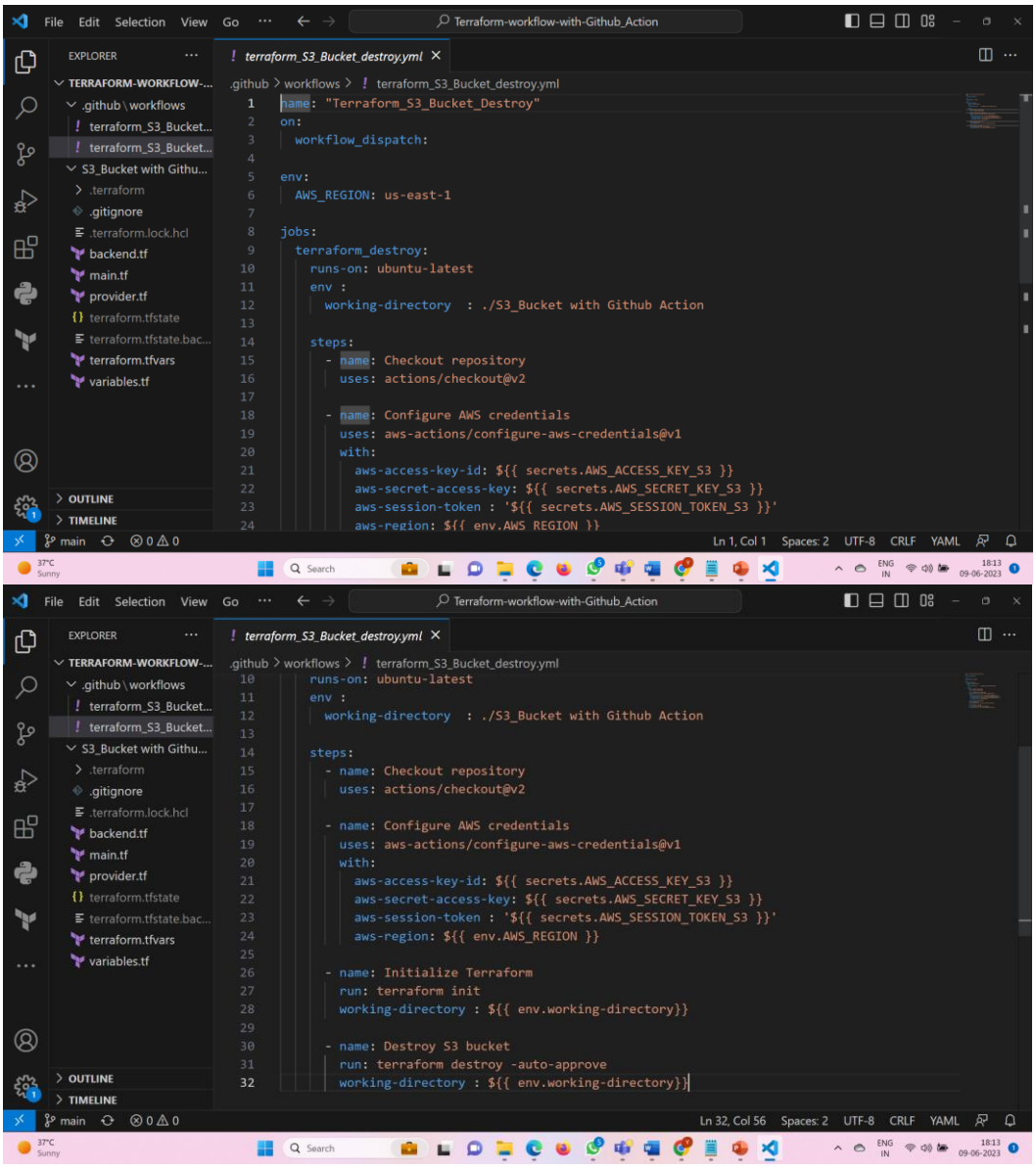
7. Now click on code and copy the link and open that .github/workflow in that directory.



8. Now that folder open in VS code and make one file for creating the terraform S3 bucket using .yml extension and write the script in VS code like that.

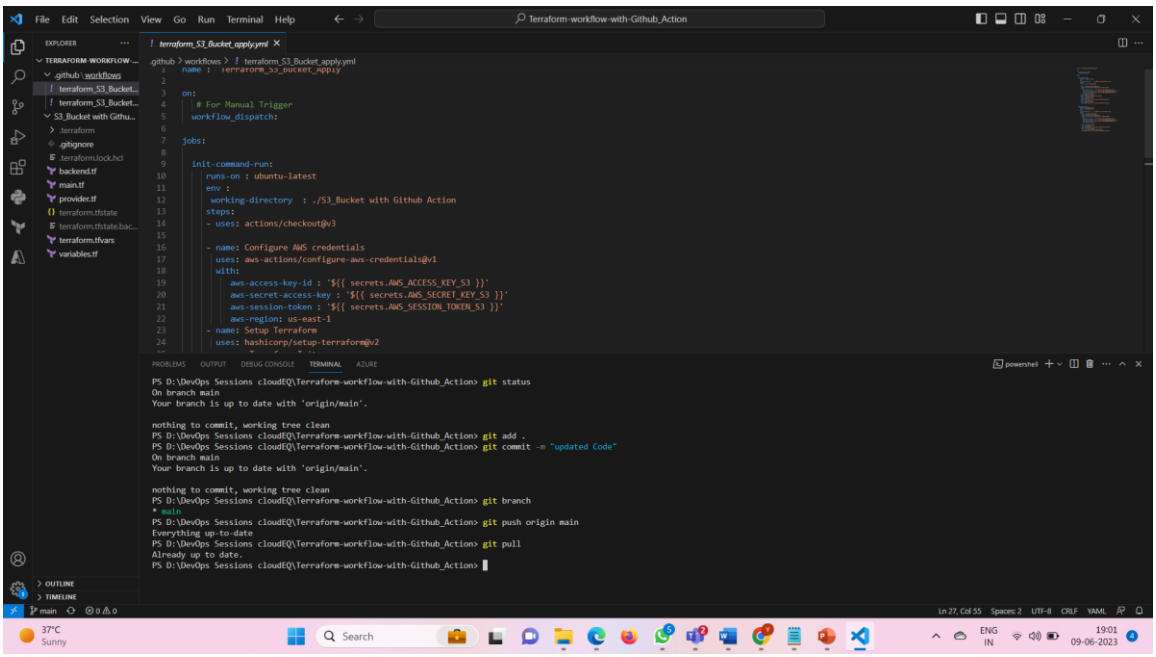


9. One another file for destroy the terraform s3 bucket using .yml extension and write the script like that.

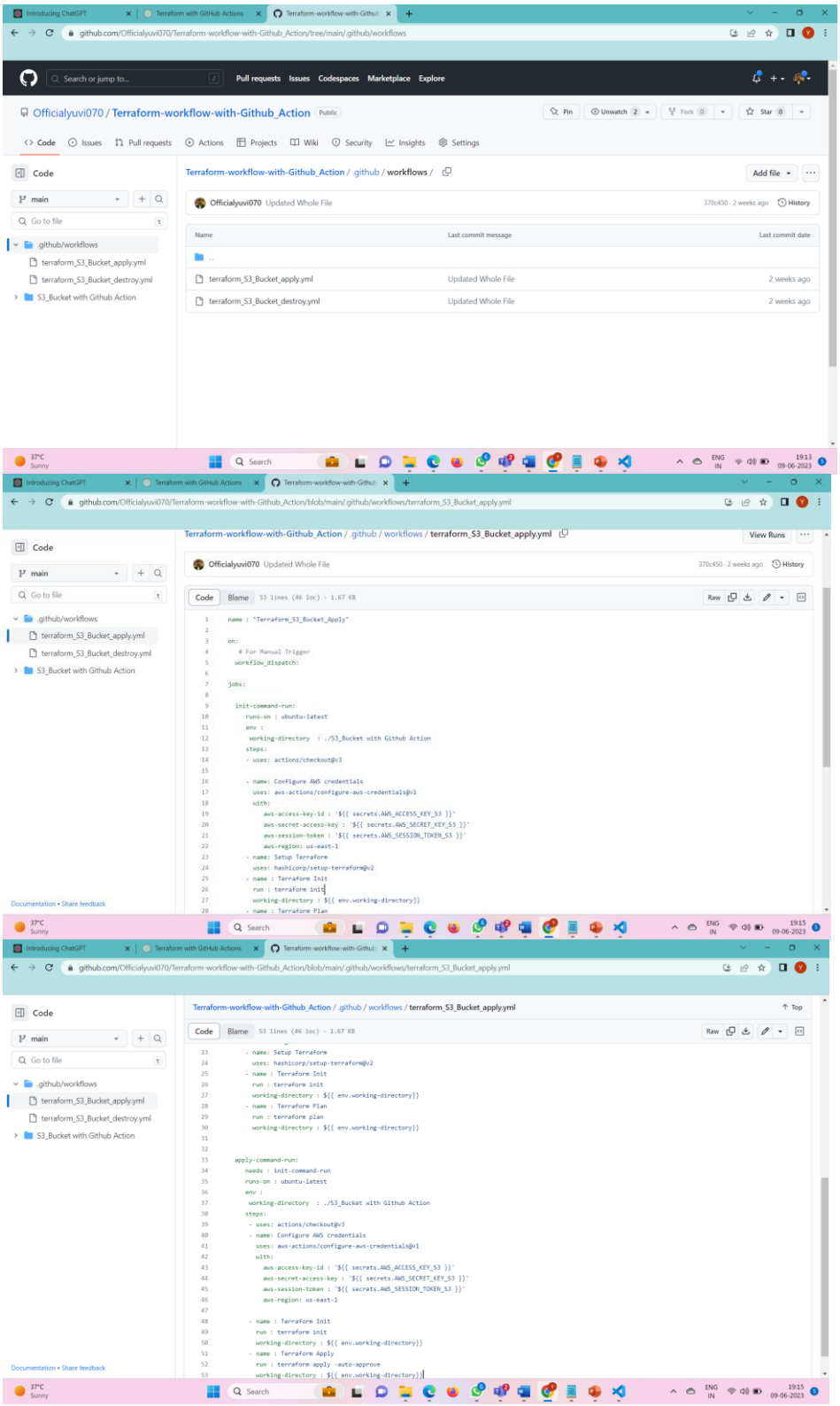


10. Now save the both script and open the Terminal and enter the commands one by one.

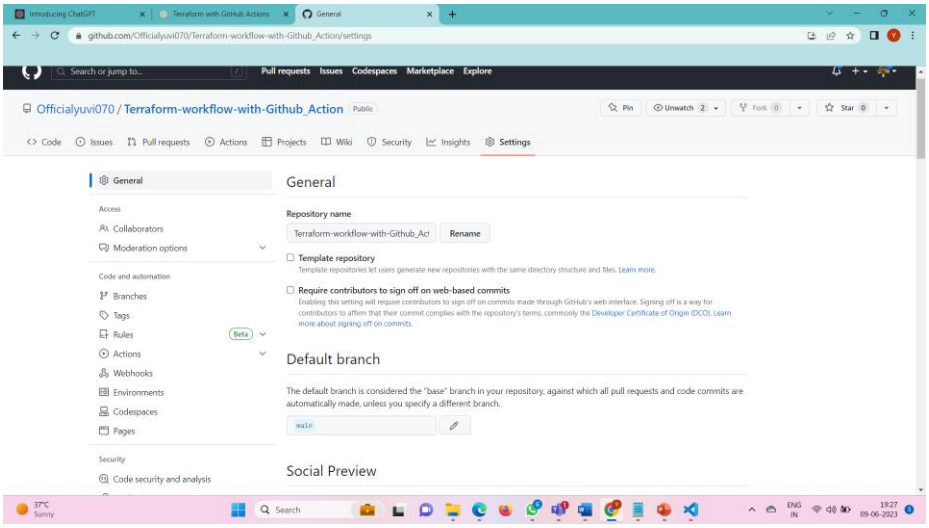
- **git status** : The git status command displays the state of the working directory and the staging area. It lets you see which changes have been staged, which haven't, and which files aren't being tracked by Git. Status output does not show you any information regarding the committed project history.
- **git add .** : The git add command adds a change in the working directory to the staging area. It tells Git that you want to include updates to a file in the next commit. However, git add doesn't really affect the repository in any significant way—changes are not actually recorded until you run git commit.
- **git commit -m "any-message"** : The git commit command is what you'll use to take all of the changes that have been made locally and push them up to a remote repository. It's important to note that you can't just type "git commit" by itself with no arguments — it needs at least one parameter, which will be either HEAD or any other branch name.
- **git branch** : The git branch command lets you create, list, rename, and delete branches. It doesn't let you switch between branches or put a forked history back together again. For this reason, git branch is tightly integrated with the git checkout and git merge commands.
- **git push origin main :** The git push command is used to upload local repository content to a remote repository. Pushing is how you transfer commits from your local repository to a remote repo. It's the counterpart to git fetch , but whereas fetching imports commits to local branches, pushing exports commits to remote branches.



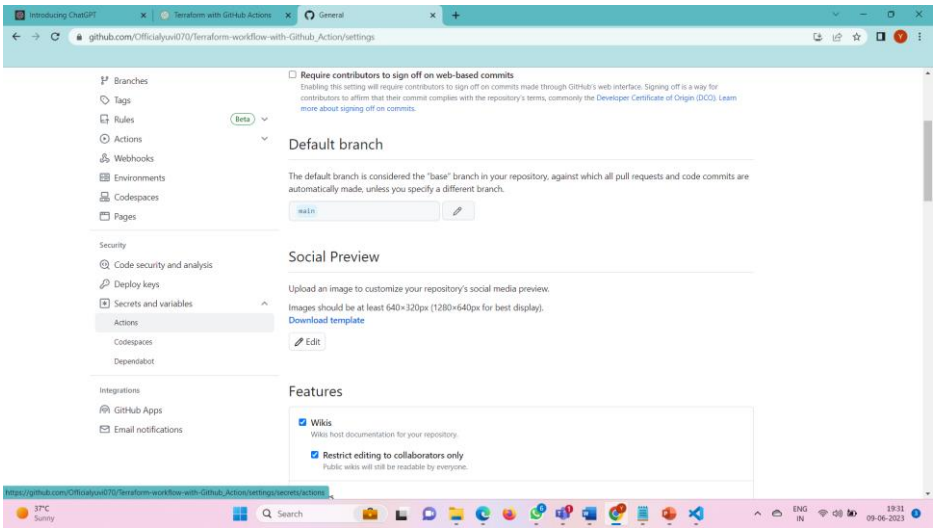
11. Now Open the GitHub profile and choose that repo and you will the code the present inside the like that.



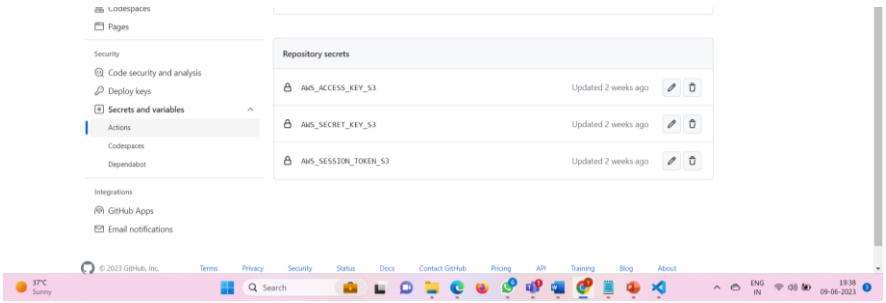
12. Now click on the **setting** in GitHub Profile.



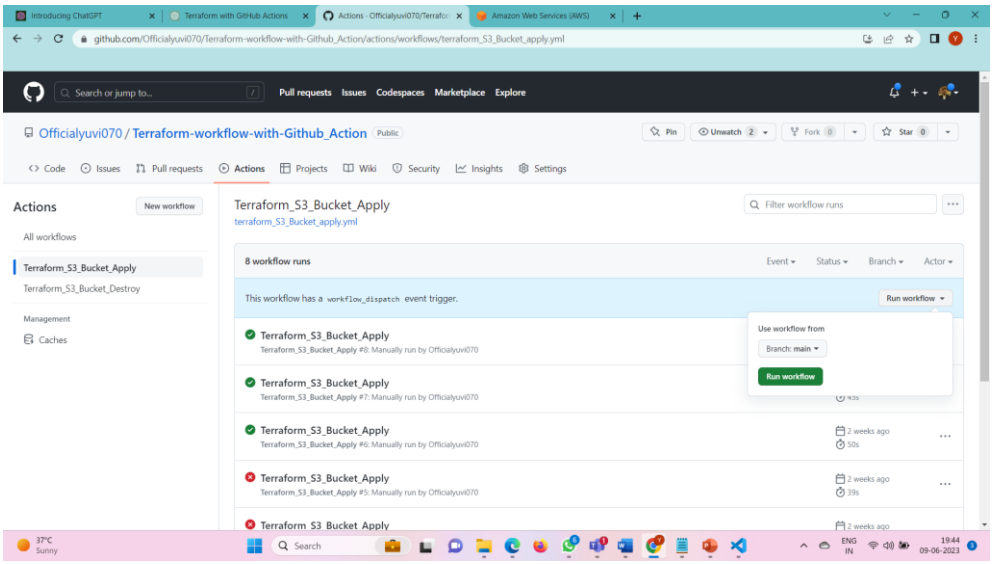
13. Now in L.H.S scroll down and click on **Secrets** and **variables** and then click on **Action** like that.



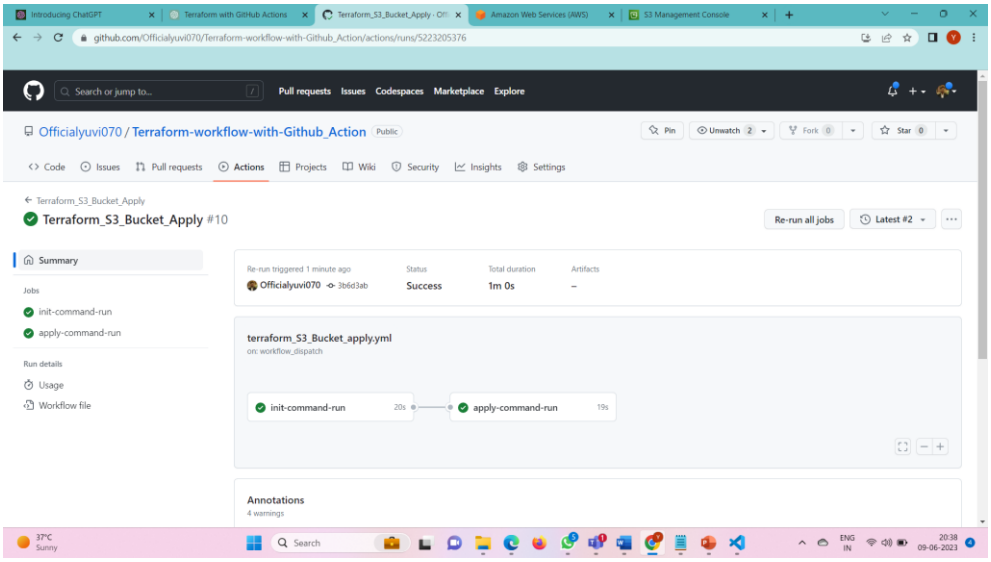
14. Here you will create the new **repository secret** for access the AWS console in which I am adding `AWS_ACCESS_KEY_S3`, `AWS_SECRET_KEY_S3` and `AWS_SESSION_TOKEN_S3` in these secrets put the values of these secrets from AMS Account command line, after adding this secrets you will see like that.



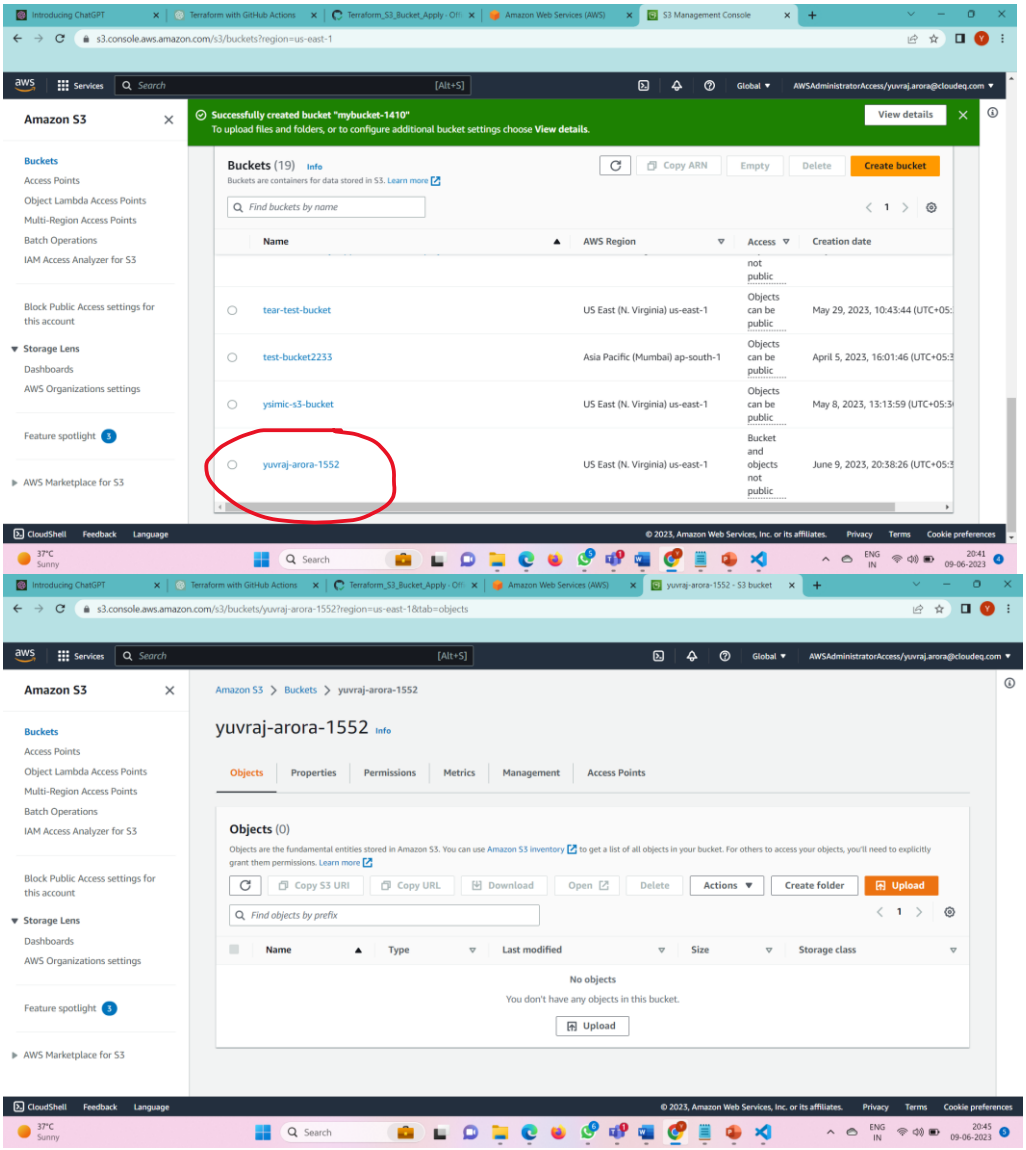
15. Now click on **Action** and then firstly click **terraform S3 bucket apply** for this next click on just **once** time **Run workflow.**



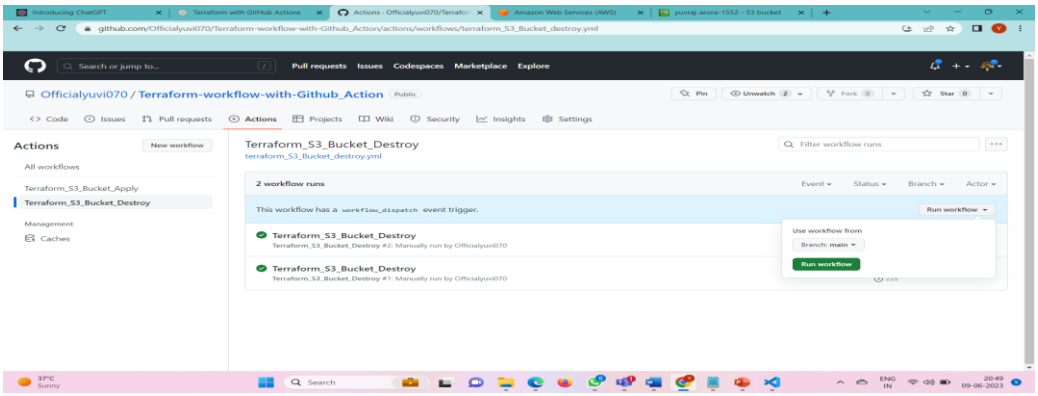
16. You will see the jobs are run automatically first init command then second apply command like that.



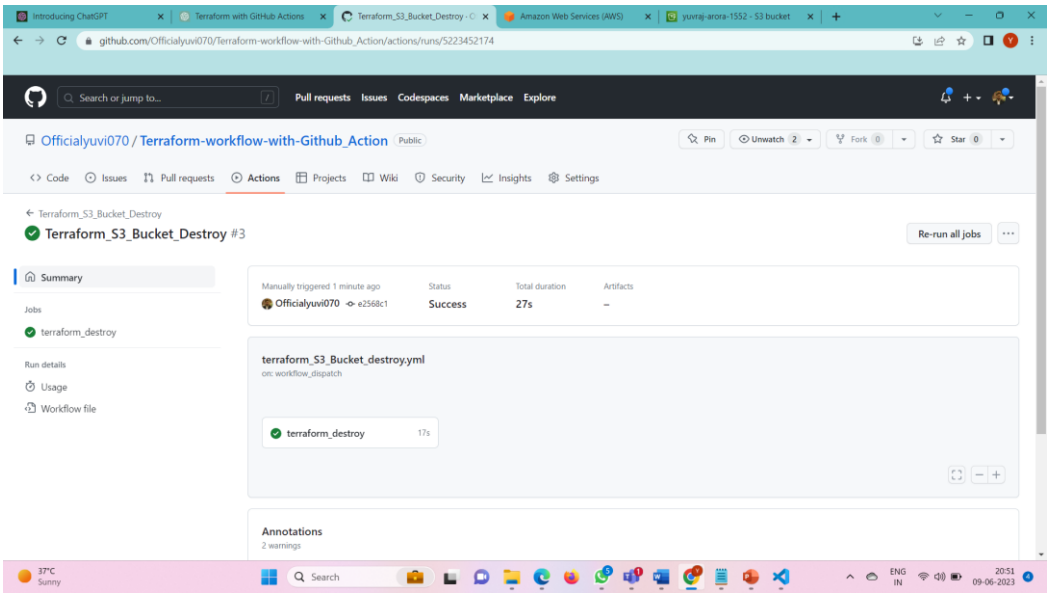
17. Now go to your AWS account and you will see the bucket is Created in S3 service.



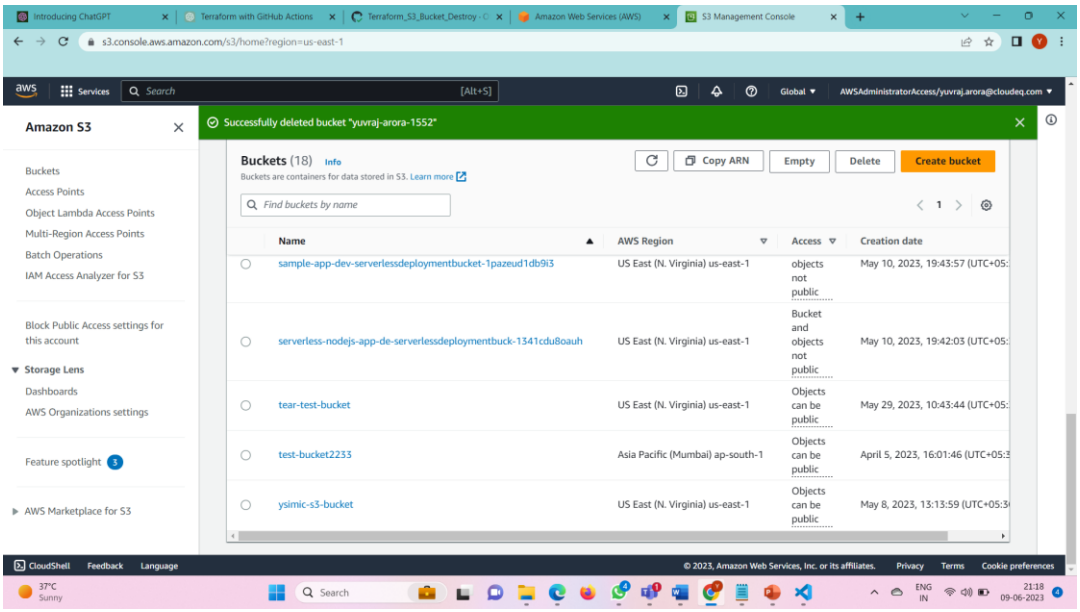
18. Now click on **Action** and then click **terraform S3 bucket destroy** for this next click on just **once** time **Run workflow.**



19. You will see the jobs are run automatically first init command then second apply command like that.



20. Now go to your AWS account and you will see the bucket is Destroy / Delete in S3 service.



Chapter 8. Conclusion and Future Scope

Conclusion :

The project "Automated Infrastructure Deployment with Terraform and GitHub Actions" has successfully achieved its objective of automating the provisioning and management of infrastructure resources using Terraform and streamlining the deployment process with the help of GitHub Actions. The project has provided a robust and efficient solution for infrastructure deployment, enabling organizations to leverage infrastructure as code and continuous integration and deployment practices. By automating the infrastructure deployment pipeline, the project has increased efficiency, reduced manual errors, and improved overall productivity.

Future Scope :

While the project has laid a strong foundation for automated infrastructure deployment, there are several avenues for future expansion and enhancement. Here are some potential areas for future development and improvement:

1. Integration with Additional Tools: The project can be extended to integrate with other complementary tools and services. For example, incorporating configuration management tools like Ansible or Chef can provide a more comprehensive solution for infrastructure configuration and management.
2. Advanced Testing and Validation: Enhancing the testing and validation capabilities can further improve the reliability and stability of the infrastructure. Implementing additional types of tests, such as security and performance testing, can ensure that the infrastructure meets the required standards and performs optimally.
3. Infrastructure Monitoring and Scaling: Integrating with monitoring and scaling tools or services can enable real-time monitoring of the deployed infrastructure, proactive issue detection, and auto-scaling based on resource utilization. This ensures the infrastructure remains healthy, responsive, and able to handle varying workloads.
4. Enhanced Collaboration and Governance: Implementing features for enhanced collaboration and governance can facilitate better teamwork and project management. For instance, introducing code review processes, access control mechanisms, and deployment approval workflows can ensure proper governance and adherence to best practices.
5. Support for Multi-Cloud Environments: Extending the project to support multiple cloud providers can enable organizations to deploy their infrastructure across different cloud platforms based on specific requirements. This enhances flexibility and reduces vendor lock-in.
6. Infrastructure Lifecycle Management: Incorporating features for managing the complete lifecycle of infrastructure resources, including provisioning, configuration changes, updates, and retirement, can provide a comprehensive solution for infrastructure management.
7. Continuous Improvement and Feedback: Implementing mechanisms for continuous improvement, such as feedback loops, retrospectives, and user surveys, can help gather insights and identify areas for further enhancement. This ensures that the project evolves based on user feedback and emerging industry trends.

References

1. Riti, Pierluigi, David Flynn, Pierluigi Riti, and David Flynn. "Terraform HCL." Beginning HCL Programming: Using Hashicorp Language for Automation and Configuration (2021): 79-105.

2. Krief, Mikael. Learning DevOps: The Complete Guide to Accelerate Collaboration with Jenkins, Kubernetes, Terraform and Azure DevOps. Packt Publishing Ltd, 2019.

3. Zampetti, Fiorella, Salvatore Geremia, Gabriele Bavota, and Massimiliano Di Penta. "Ci/cd pipelines evolution and restructuring: A qualitative and quantitative study." In 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 471-482. IEEE, 2021.

4. Valenzuela-Toledo, Pablo, and Alexandre Bergel. "Evolution of GitHub Action Workflows." In 2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 123-127. IEEE, 2022.

5. Anderson, Charles. "Docker [software engineering]." *Ieee Software* 32, no. 3 (2015): 102-c3.

6. Dillon, Tharam, Chen Wu, and Elizabeth Chang. "Cloud computing: issues and challenges." In *2010 24th IEEE international conference on advanced information networking and applications*, pp. 27-33. Ieee, 2010.

7. Saeed, Iman, Sarah Baras, and Hassan Hajjdiab. "Security and privacy of AWS S3 and Azure Blob storage services." In 2019 IEEE 4th International Conference on Computer and Communication Systems (ICCCS), pp. 388-394. IEEE, 2019.

8. Wang, Guohui, and TS Eugene Ng. "The impact of virtualization on network performance of amazon ec2 data center." In 2010 Proceedings IEEE INFOCOM, pp. 1-9. IEEE, 2010.

9. Arvindhan, M., and Abhineet Anand. "Scheming an proficient auto scaling technique for minimizing response time in load balancing on Amazon AWS Cloud." In International Conference on Advances in Engineering Science Management & Technology (ICAESMT)-2019, Uttaranchal University, Dehradun, India. 2019.

10. Mathew, Sajee, and J. Varia. "Overview of amazon web services." Amazon Whitepapers 105 (2014): 1-22.

11. Beach, Brian, Steven Armentrout, Rodney Bozo, Emmanuel Tsouris, Brian Beach, Steven Armentrout, Rodney Bozo, and Emmanuel Tsouris. "Virtual private cloud." Pro Powershell for Amazon Web Services (2019): 85-115.

12. Liao, Wen-Hwa, Ssu-Chi Kuai, and Yu-Ren Leau. "Auto-scaling strategy for amazon web services in cloud computing." In 2015 IEEE International Conference on Smart City/SocialCom/SustainCom (SmartCity), pp. 1059-1064. IEEE, 2015.

13. Soni, Ravi Kant, Namrata Soni, Ravi Kant Soni, and Namrata Soni. "Deploy MySQL as a Database in AWS with RDS." Spring Boot with React and AWS: Learn to Deploy a Full Stack Spring Boot React Application to AWS (2021): 77-102.